Analysis and Development of Quorum Protocols for Real-World Network Topologies

Fakultät II - Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik der Carl von Ossietzky Universität Oldenburg zur Erlangung des Grades und Titels eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

angenommene Dissertation von Herrn Robert Schadek
geboren am 02.07.1986 in Rotenburg/Wümme

Gutachter: Prof. Dr.Ing. Oliver Theel
Weitere Gutachter:
Prof. Dr. Oliver Kramer

Tag der Disputation: 29.03.2021

for Nele

Abstract

English

Highly available services can be implemented by means of so-called quorum protocols. Unfortunately, applying quorum protocols in real-world physical networks turns out to be difficult since efficient quorum protocols often depend on a particular graph structure imposed on the replicas managed by it. In this work, we show that the cost and availability predictions of the operations provided by quorum protocols are often not accurate when they are executed in a real-world physical network. We present the mapping approach, the mapping approach increases the accuracy of the cost and availability predictions of most quorum protocols on real-world networks. The mapping approach is used to analyze multiple existing quorum protocols when applied on real-world networks. As the mapping approach is computational expensive, the k-nearest neighbors algorithm is novelly employed to predict availability of the services facilitated by the quorum protocols when used on a real-world network. As even this technique is infeasible at certain network sizes two new quorum protocol, namely the Circle Protocol and the Crossing Protocol, are presented that directly work on the real-world network of arbitrary size. All these different techniques are extensively analyzed and compared. Concluding, an algorithm is presented that uses all these techniques to find the best quorum protocol for a given real-world network.

German

Hochverfügbare Dienste können mit Hilfe von sogenannten Quorum Protokollen realisiert werden. Leider gestaltet sich die Anwendung von Quorum-Protokollen in realen Netzwerken als schwierig, da effiziente Quorum-Protokolle oft von einer bestimmten Graphenstruktur abhängen, die den von ihnen verwalteten Replikaten auferlegt sind. In dieser Arbeit zeigen wir, dass die Kosten- und Verfügbarkeitsvorhersagen von Quorum-Protokollen oft ungenau sind, wenn sie in einem realen Netzwerk eingesetzt werden. Wir stellen den Mapping-Ansatz vor. Der Mapping-Ansatz erhöht die Genauigkeit der der Kosten- und Verfügbarkeitsvorhersagen der meisten Quorum-Protokolle in realen Netzwerken. Da der Mapping-Ansatz sehr rechenintensiv ist, wird das k-nearest neighbors Verfahren benutzt, um die Verfügbarkeit der durch die Quoren Protokolle erbrachten Dienste vorherzusagen. Da selbst diese Technik bei bestimmten Netzwerkgrößen nicht praktikabel ist, werden zwei neue Quorum-Protokoll, namentlich das Circle-Protokoll und das Crossing-Protokoll, vorgestellt. Diese beiden Protokolle arbeiten direkt auf einem realen Netzwerk. Alle diese verschiedenen Techniken werden ausführlich analysiert und

verglichen. Abschließend wird ein Algorithmus vorgestellt, der alle diese Techniken verwendet um das beste Quorum-Protokoll für ein gegebenes reales Netzwerk zu finden.

Contents

1 Introduction

Computer fail from time to time. If a piece of data is stored on this failed computer, the stored data is no longer accessible. To increase the availability of that data, it can be replicated to multiple computers. As long as one of the computers, storing a replica, is available the data is accessible. Another advantage of replication is the locality of the accessed data. Assuming the computers, storing the replicas, are networked. Then accessing the data from a computer near the user is likely faster or cheaper, due to the physical proximity, than accessing the data from a further away located computer.

These replicas need to be managed. Quorum protocols (QPs) are one way to manage these replicas. Generally, QPs offer a read and a write operation. One of the things to manage is the synchronization between the replicas and between the operations. Let the piece of data or data object (DO) be replicated on five replicas, as shown in Figure 1.1. If the DO located on replica 1 is written with a

Figure 1.1: Five replicas storing a DO.

new value, then reading the DO from replica 4 does not yield the last written value. For most applications, this is not the intended behavior [1]. Additionally, without synchronization two write-operation could potentially write different values at the same time. For example, the value a is written to the DO on replica 2 and the value b is written to the DO on replica 3. In the absence of synchronization, it is unclear whether a or b is the last written value. Usually, the goal is that both read and write operations behave as they would do on a non-replicated DO. This means that even though possible executed in parallel, the history of the data will appear as it would after being executing in a serial manner. This non-replicated behavior can be achieved by control protocols that guarantees the one-copy serializability (1SR) property [2]. Many QPs implement such a control protocol.

The QPs presented in this work manage a static number of replicas. For each operation, a set of replicas is identified that executes the operation. These sets of replicas are called quorums and consist of a subset of the replicas managed by the QP. Often, two kinds of quorums exist. One kind of quorum read operations called read quorum (RQ), and another kind for write operations called write quorum (WQ).

Quorums have a wide range of applicability. They have been used, for example, for managing data replication (DR) and mutual exclusion [3, 4, 5]. This work will focus on QPs used for data replication.

A common way for QPs to achieve 1SR is to construct the WQs in a way that they have at least one replica in common with every other WQ:

$$\forall q, q' \in QW, q \neq q' : q \cap q' \neq \emptyset. \tag{1.1}$$

This way, e.g. locking a WQ for writing, prevents every other write operation from locking an additional WQ, as a replica has only one write lock. Additionally, replicas contained in a RQ are locked for reading. Furthermore, every RQ is constructed in a way such that at least one replica of the RQ is also part of every WQ:

$$\forall q \in QR, q' \in QW : q \cap q' \neq \emptyset \tag{1.2}$$

To execute a read (write) operation, all replicas in the RQ (WQ) have to be locked for the desired operation and any replica can only be locked for one operation at a time. This requirement allows a RQ to always read the most recently written data, identified by timestamps or version numbers (VNs), as at least one replica of every RQ has been part of the last written WQ.

As an example, consider a WQ consisting of three out-of five replicas of Figure 1.1. A write operation writes the value c with a VN of 5 to a WQ consisting of the three replicas $0, 1$, and 3. Figure 1.2 shows the three selected replicas enclosed in a red circle. The lower index represents the VN and the character above the VN represents the DO. Figure 1.3 shows the replicas after the write operation has

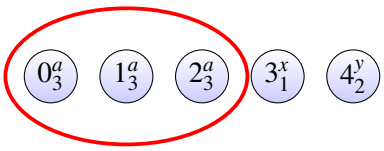


Figure 1.2: Three replicas being selected to form a WQ to execute a write operation.

been executed. It can be seen that the values, of the three selected replicas, have changed to c and that their VN is set to 4. Following that, a read operation is

Figure 1.3: Five replicas after writing the value c with VN 4 to the replicas $0, 1$, and 2.

executed that uses the three encircled replicas, as shown in Figure 1.4, as a RQ. It can be seen that the one replica that took part in the write operation also takes

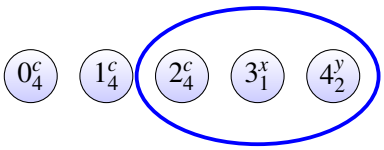


Figure 1.4: Three replias being selected to execute a read operation.

part in the read operation. Figure 1.5 displays this visually. The VNs of the three

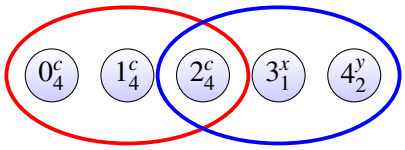


Figure 1.5: Showing the intersection between a RQ and a WQ.

read replicas are compared. By definition, the DO with the highest VN is the last written data. This example is mostly equivalent with the Majority Consensus Protocol (MCS). The MCS is presented in more detail in Section 4.1 on page 25.

So far, it has not been discussed how the replicas communicate. In the shown example, it is assumed that every replica can directly[1] communicate with every other replica used in the QP as shown in Figure 1.6. If a classical network topology, as shown in Figure 1.7, would be used for the communication between the five replicas, the problem becomes obvious. When the replica with ID 2 becomes unavailable, no RQ or WQ can be formed anymore, as two unconnected partitions of two replicas remain. The assumption that every replica can directly communicate with ever other replicas is implicitly taken by most QPs.

This is a very strong assumption, as the QP usually has no influence on the network topology it uses. This work will show that the actual communication structure, which is called physical network topology (PNT) in this work, has to be considered in the availability and cost analysis of a QP.

To do that, a system model is presented in Chapter 2 on page 7 that is used throughout this work. After that, a more rigorous introduction to data replication and QPs is given in Chapter 3 on page 11. Chapter 4 on page 25 presents three common and well-known QPs that will be used later for in-depth analysis. In Chapter 5 on page 45 a technique is presented that can be used to analyze the impact any specific PNT has on the availability and costs measurements of any QP. Here, it is shown that the PNT used, is an important factor to consider during the analysis of a QP. Based on the insights gained during these analyses two new QPs are developed and presented in this work, that directly use a given

[1] By directly, a point-to-point connection between two replicas is assumed.

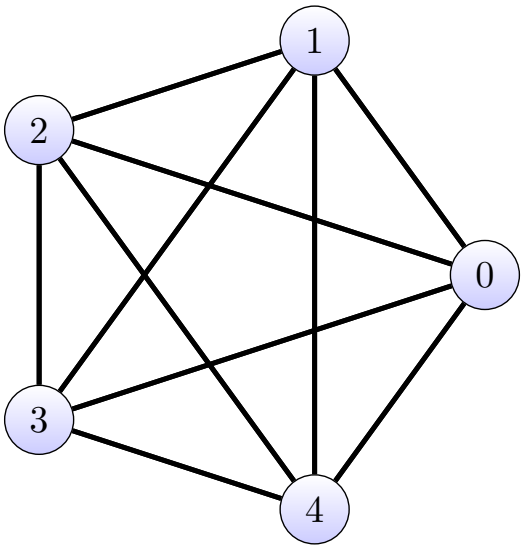


Figure 1.6: The graph structure (GS) used by the MCS in the example with five replicas.

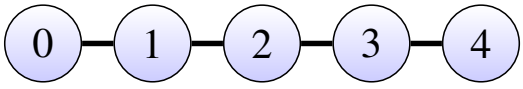


Figure 1.7: A GS that is easy to partition.

PNT, at thereby try to avoid the overhead of an additional indirection. Both new QPs directly use the provided PNT as means for the communication. The first new QP is called Circle Protocol (CIP) and is presented in Chapter 6 on page 131. The second new QP, named Crossing Protocol (CP), is presented in Chapter 7 on page 165. In Chapter 8 on page 193 an algorithm is presented that allows to decide which QP to use on a given PNT to achieve the highest operation availability or the lowest operation costs. Many of the analyses presented in this work are so computationally complex that even with modern computers their straightforward analysis would take an unreasonable amount of time, in the order of years. Therefore, a significant effort was invested to optimize the execution performance of the developed analysis tools. Some of this work is presented in Chapter 9 on page 197. A conclusion of this work is given in Chapter 10 on page 231. In Chapter 11 on page 233, ideas for future works are presented.

2 System Model

Before the different QPs and research questions can be discussed a common language and notation needs to be established, this is one in this chapter.

2.1 Set

A set is a collection of unique elements. $S = \{1,2,3\}$ shows a set S containing the numbers $1,2$, and 3. The elements of a set are not ordered in any way. Let S be set, then the notation $|S|$ states the number of elements in it. Let $S = \{1,2,3\}$ be a set, then the notation $1 \in S$ states that the number 1 is an element of the set S.

The expression $S' = \{x | x \in \mathbb{N} \land x \bmod 2 = 0 \land x > 0 \land x < 10\}$ defines a set S' with the help of the so called set-builder notation. The set-builder notation consists of four parts. The first part is the variable. In the shown example, the variable has the name x. The variable is followed by | which separates the variable and the restrictions imposed on the values that make up the set. The first restriction in the example is the domain of the values. The domain of the example is $\mathbb{N}$. The domain is sometimes omitted, but this may lead to a problem known as the Russell paradox [6]. Therefore, the domain for all set-builder expressions will always be stated. The last part is the logical predicate. If the predicate is true for the value held by the variable, then the value is added to the constructed set. The above expression is read as follows: From all values in $\mathbb{N}$ create a set where each element is greater than 0, smaller than 10, and even. The set S' consists of the elements $\{2,4,6,8\}$.

Let S be a set, then an expression such as $x \in S$ states that x is an element of the set S. Let S be a set, then an expression such as $x \notin S$ states that x is not an element of the set S. The symbol $\emptyset$ denotes the empty set.

Let $S1$ and $S2$ be two sets, then $S1 \cup S2 = \{x | x \in S1 \lor x \in S2\}$, where the operator $\cup$ is called the set union operator [7].

Let $S1$ and $S2$ again be two sets then $\cap$ is known as the set intersection operator where $S1 \cap S2 = \{x | x \in S1 \land x \in S2\}$ [7].

2.2 Tuple

A tuple is an ordered finite sequence of elements. $T = (a,b)$ is a tuple of two elements named a, and b. $H = ()$ is an empty tuple, which contains no elements. If the elements of the tuple are named, then the elements of the tuple can be accessed by the T_{name} notation. For example, T_a accesses the element a of the

tuple T. If the elements of the tuple are unnamed, the elements of the tuple are accessed by an index. The first element of a tuple is indexed by 1, the second by 2 and so on. For example, consider $T' = (\{a,b,c\},\{1,2,3\})$, the first element of the tuple $\{a,b,c\}$ is accessed with the notation T'_1. Let T be a tuple, then the notation $|T|$ states the number of elements in it.

2.3 Graph Structure

A GS $G = (V,E)$ is a two-tuple of a set of vertices V and a set of edges E. One edge connects two vertices. G_V gives the set of vertices of a GS. G_E gives the set of edges of a GS. A vertex $v \in V$ is a three tuple $v = (i, c_x, c_y)$, where $i \in \mathbb{N}$ is the ID of the vertex and c_x and c_y are the coordinates (i. e. its position/location in a plane) of the vertex in the corresponding dimensions. Vertex IDs are unique for a given GS G, therefore:

$$\forall x, y \in G_V : x_i \neq y_j \tag{2.1}$$

is true. The shorthand notation v_i gives a vertex with ID i.

An edge $e_{i,j} \in G_E$ is defined as $e_{i,j} := (v_i, v_j)$, where $(v_i, v_j) \in G_V : i \neq j$.

A tuple $a = (v_1, v_2, \ldots, v_n)$ is a path between v_1 and v_n in graph G, iff:

$$\forall i, 1 \leq i \leq n : \exists v_i \in G_V \text{ and} \tag{2.2}$$
$$\forall i, 1 \leq i < n : \exists e = (v_i, v_{i+1}) \in G_E \tag{2.3}$$

If a path exists, then the two vertices v_1 and v_n are called connected. $\mathbb{V}(v_1, v_2, \ldots, v_n)$ denotes the set of vertices of a path such that:

$$\mathbb{V}(v_1, v_2, \ldots, v_n) := \{v_1, v_2, \ldots, v_n\}. \tag{2.4}$$

$\mathbb{E}(v_1, v_2, \ldots, v_n)$ denotes the set of the edges of a path such that:

$$\mathbb{E}(v_1, v_2, \ldots, v_n) := \{e_{1,2} \ldots, e_{n-1,n}\}. \tag{2.5}$$

The shorthand notation $\Omega(v_i, v_j) \in G$ test if there exists a path between vertices v_i and v_j in GS G. $\Psi(v_i, v_j) \in G$ is the shorthand notation to get a tuple that represents the shortest path between the vertices v_i and v_j in GS G. If the vertices v_i and v_j are not connected in G, then an empty tuple () is obtained.

A path $p = \langle v_i, \ldots v_j \rangle \models \Omega(i,j) \in \text{GS}$ is considered to partition a GS into two GSs $GS_1 = (V_1, E_1), GS_2 = (V_2, E_2)$ iff, given p:

$$\begin{aligned} \nexists p' : & \Omega(v, v') \in \text{GS} \models p' \wedge p \neq p' \\ & \wedge v \in V_1 \wedge v' \in V_2 \\ & \wedge \mathbb{V}(p') \cap \mathbb{V}(p) = \emptyset. \end{aligned} \tag{2.6}$$

This basically states that a path partitions a GS, if there is no path from GS_1 to GS_2 whose vertices do not intersect with the vertices of the partitioning path.

3 Data Replication

The idea behind DR is to increase the availability of the stored DOs by creating multiple replicas of it. If one replica is no longer available, another replica might still be accessible. The goal of a data replication system (DRS) is to provide highly available and low cost read and write access to the DO. DR is a branch of distributed system (DS).

DRS are often classified into four subcategories as shown in Table 3.1. Syntactic

syntactic	pessimistic syntactic DRS	optimistic syntactic DRS
semantic	pessimistic semantic DRS	optimistic semantic DRS
	pessimistic	optimistic

Table 3.1: Classification of Data Replication Systems.

DRS treats all data the same, which means that syntactic DRS do not use specific properties of the stored data to increase the availability or to decrease the costs of the operations accessing the DO. Semantic DRS on the other hand exploit specific properties of the stored data to increase availability or to decrease costs. This work only considers syntactic DRS [8].

Pessimistic DRS provide strict consistency properties, such as the 1SR property. Optimistic DRS provide less strict consistency properties than pessimistic DRS, generally allows for operations with higher availability but may lead to inconsistencies in the stored data. This work only considers pessimistic syntactic DRS.

3.1 Data Replication System Model

A DRS consists of one or more replicas. A replica stores a DO. The replicas of a DRS are independent of each other. This means, replicas may fail independently of each other. Replicas, in this work, exhibit fail-stop behavior [9]. Replicas can use messages to communicate with each other. The availability of a replica is given by p, where

$$p \in \mathbb{R} : p \in [0,1]. \tag{3.1}$$

The higher the value p, the higher the availability of the replica. Availability is the probability that the replica has not failed. All replicas of a DRS are assumed to have the same p-value.

One replica is hosted by one vertex of this GS. The edges of the GS serve as communication channels between the replicas. The topology of the communication channels between the replicas is defined by a GS. The communication channels between replicas are always available and are modeled by the edges of a GS. The presumed strength of this assumption, is diminished by the fact that

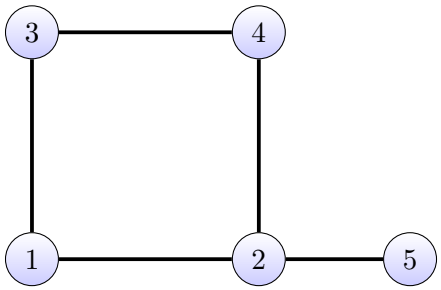


Figure 3.1: An example of a GS serving as a communication network in a DRS consisting of five replicas.

communication channel availabilities other than 1 can be at least approximated by decreasing the replica availability accordingly. I.e. consider the GS in Figure 3.2. Assuming, that communication channel has an availability of 0.8, and the replicas

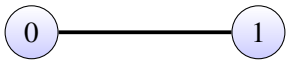


Figure 3.2: An example to justify always available communication channels/edges.

0 and 1 have a p value of 0.9 the probability that the replicas can communicate, or system availability, is $0.9^2 \cdot 0.8 = 0.648$. The approximately same system availability, under the assumption that the communication channel is available, can be achieved by by decreasing the replica availability to $p = 0.805$. This results in $0.805^2 \cdot 1 \approx 0.648$. The second reason, that communication channel/edge are assumed to have an availability of $p = 1$, is that the complexity of the analyses of QPs increases rapidly with in increase in the number of replicas. Such much so, that a complete chapter of this work was dedicated to manage this complexity. Also considering communication channel availability in the analyses would very likely make the problem unmanageable.

3.2 One-copy serializability

Using multiple replicas increases the availability of the DO as it can often be accessed using different replicas. But replicating the DO introduces the need for synchronization as demonstrated in the introduction. Let the DO be replicated on five replicas, as shown in Figure 3.3. If the DO located on replica 0 is updated with a new value, followed by a read of the DO stored on replica 4, replica 4 does

Figure 3.3: Five replicas of a DO.

not yield the up-to-date value. Usually, this is not the intended behavior of a read operation. An additional problem is that two concurrent write operations can be executed on different replicas of the same DO at the same time. For example, value *a* is written to the DO on replica 2 and value *b* is written to the DO on replica 3. This raises the following question: Which value is the correct one? Both examples show that simply creating multiple replicas of a DO does not necessarily lead to the expected behavior. Usually, the goal is that all operations behave as they would do on a non-replicated DO. More formally, this non-replicated behavior can be achieved by a control protocol that guarantees 1SR [2].

3.2.1 Version numbers (VNs) and Locking

VNs are an effective way to identify up-to-date DOs stored on replicas. Given a set of DOs, the DOs with the highest VN is considered up-to-date.

Each replica allows the execution of multiple read or a single write operation on its stored DO, as read and write operation on the same DO cannot be executed simultaneously. Therefore, replicas offer two kinds of locks. A read-lock and a write-lock. If a replica is locked for reading, it cannot be locked for writing. If it is locked for writing, it cannot be locked for reading. Many processes can lock a replica for reading, but only one process can lock a replica for writing.

3.2.2 1SR Example

The following example shows how VNs and locks can be used to specify a read and write protocol that adheres to 1SR. To demonstrate that, the example will first show how a write operation is executed. After that, a read operation is executed that will show how the last written data is read.

Figure 3.4 shows the initial state of the DS consisting of five replicas. The task of the DRS only handles a single DO. The replicas are named $0, 1, 2, 3$, and 4. The lower case letter next to the ID is the VN of the DO stored on the replica. E. g. the DO stored by replica 0 has the VN 3. In Figure 3.5 on the next page, three

Figure 3.4: Five replicas of a DO with VNs.

replicas are identified, known as the WQ, to execute the write operation. These

three replicas 0, 1, and 3 are encircled in the red circle. These three replicas are then locked for writing. The DO is then written onto the replicas and the VN is incremented. The new VN *nv* is calculated as:

$$vn = maxVN + 1 \tag{3.2}$$

where *maxVN* is the highest VN of the replicas locked for writing. As shown in Figure 3.6 the new highest VN is 4. The last part of this example is to show a

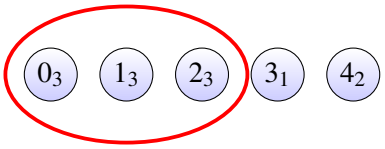


Figure 3.5: Three out of five replicas are selected for a write operation.

read operation. The three replicas locked for reading are shown in Figure 3.7. The replicas 2, 3, and 4 are encircled in the blue circle are the replicas used for reading. The read operation identifies that replica 2 has the highest VN. Therefore, the DO

Figure 3.6: Three out of Five replicas after the execution of the write operation.

stored by replica 2 is the last written one, which in turn is then read. The strategy

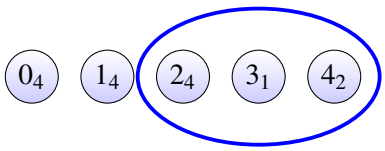


Figure 3.7: Three out of Five replicas selected for a read operation.

implicitly described in this example is one-copy serializable. As a majority of the replicas are locked by any write operation, no two concurrent write operations are executable. Also, a majority of the replicas need to be locked for reading, this prevents read operations to be executed concurrently with write operations. Additionally, the described use of the VN ensures that the last written data is read.

This example, partially describes the strategy used by the MCS to achieve 1SR. A more detailed description of the MCS is given in Section 4.1.

3.3 Quorum Protocols

The QPs covered in this work implement a control protocol that provides a read and a write operation on a DO. The QPs covered are pessimistic-syntactic DRSs and provide 1SR for their operations.

QPs have a set of N replicas assigned to them. QPs use quorum to execute their read and write operations. A quorum is a subset of the N replicas of the QP. A RQ execute a read operation. A WQ execute a write operation.

3.4 Read- and Write-Availability and Costs

The performance of QPs is commonly measured by four criteria. These four criteria are the:

$a_r(p)$**:** the read operation availability of a for a QP when the availability of its replicas is defined by p,

$a_w(p)$**:** the write operation availability of a for a QP when the availability of its replicas is defined by p,

$c_r(p)$**:** the read operation costs of a for a QP when the availability of its replicas is defined by p,

$c_w(p)$**:** the write operation costs of a for a QP when the availability of its replicas is defined by p.

The availability of an operation is measured by a $p \in [0,1]$. The higher the p-value, the higher the availability of the operation. The cost of an operation is measured by the average minimal number of replicas required to execute the operation. An operation that requires fewer replicas is cheaper than those that require more replicas. This work only considers the number of replicas in the cost analyses. It does not consider the number of messages send between the replicas, which would be needed if a commit protocol, like the two-phase commit protocol [8, 2], is to be analyzed.

Some QPs have a closed formula to compute these values. But unfortunately, this is not the case for all QPs.

The semantic-pessimistic QPs considered in this work all use a set of replicas to execute their operations. All of these QPs have a read and a write operation. It is assumed that QPs, and this is true for the QP presented in this work, allow to test whether a subset of replicas managed by them is a RQ, or a WQ which can be used to execute the read or write operation of the Grid Protocol (GP). The expressions

$$\mathit{isReadQuorum}(q, Q) \text{ and} \tag{3.3}$$
$$\mathit{isWriteQuorum}(q, Q) \tag{3.4}$$

are used to donate these tests. q is a quorum that is tested and Q is the QP. If q is a RQ under Q $isReadQuorum(q,Q)$ evaluates to true. If q is a WQ under Q $isWriteQuorum(q,Q)$ evaluates to true. Otherwise, both evaluate to false.

Let the notation $|Q|$ donate the set of replicas of a QP Q. $\mathfrak{P}$ describes the power-set of a given set [10].

The following equations shows how *isReadQuorum* and *isWriteQuorum* can be used to construct the read operation availability ($a_r(p)$), the write operation availability ($a_w(p)$), the read operation cost ($c_r(p)$), and write operation cost ($c_w(p)$)for a given QP.

$$\begin{aligned}
\text{RQS} = \{&(q_1, \{sq_{1,1}, \ldots, sq_{1,m}\}), \ldots, (q_n, \{sq_{n,1}, \ldots, sq_{n,z}\}) && (3.5)\\
&\mid q_i \in \mathfrak{P}(|Q|) && (3.6)\\
&\wedge sq_i \in \mathfrak{P}(|Q|) && (3.7)\\
&\wedge isReadQuorum(q_i, Q) && (3.8)\\
&\wedge (q_i, sq_{i,j}) : sq_{i,j} \supset q_i && (3.9)\\
&\wedge q_i, q_j : q_i \not\supseteq q_j && (3.10)\\
&\wedge (q_i, sq_{i,n}), (q_j, sq_{j,m}) : sq_{i,n} \neq sq_{j,m} && (3.11)\\
&\wedge \nexists (q_i, sq_{i,j}), q_j : sq_{i,j} \supset q_j \wedge |q_i| > |q_j| && (3.12)\\
\}&
\end{aligned}$$

$$a_r(p) = \sum_{\forall (q,sq) \in RQS} p^{|q|}(1-p)^{N-|q|} + \sum_{\forall t \in sq} p^{|t|}(1-p)^{N-|t|} \quad (3.13)$$

$$c_r(p) = \frac{\sum_{\forall (q,sq) \in RQS} |q|(p^{|q|}(1-p)^{N-|q|}) + \sum_{\forall t \in sq} |q|(p^{|t|}(1-p)^{N-|t|})}{a_r(p)} \quad (3.14)$$

$$\begin{aligned}
\text{WQS} = \{&(q_1, \{sq_{1,1}, \ldots, sq_{1,m}\}), \ldots, (q_n, \{sq_{n,1}, \ldots, sq_{n,z}\}) && (3.15)\\
&\mid q_i \in \mathfrak{P}(|Q|) && (3.16)\\
&\wedge sq_i \in \mathfrak{P}(|Q|) && (3.17)\\
&\wedge isWriteQuorum(q_i, Q) && (3.18)\\
&\wedge (q_i, sq_{i,j}) : sq_{i,j} \supset q_i && (3.19)\\
&\wedge q_i, q_j : q_i \not\supseteq q_j && (3.20)\\
&\wedge (q_i, sq_{i,n}), (q_j, sq_{j,m}) : sq_{i,n} \neq sq_{j,m} && (3.21)\\
&\wedge \nexists (q_i, sq_{i,j}), q_j : sq_{i,j} \supset q_j \wedge |q_i| > |q_j| && (3.22)\\
\}&
\end{aligned}$$

$$a_w(p) = \sum_{\forall (q,sq) \in WQS} p^{|q|}(1-p)^{N-|q|} + \sum_{\forall t \in sq} p^{|t|}(1-p)^{N-|t|} \quad (3.23)$$

$$c_w(p) = \frac{\sum_{\forall (q,sq) \in WQS} |q|(p^{|q|}(1-p)^{N-|q|}) + \sum_{\forall t \in sq} |q|(p^{|t|}(1-p)^{N-|t|})}{a_w(p)} \quad (3.24)$$

A read quorum set (RQS) and write quorum set (WQS) are used to evaluate the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and $c_w(p)$. RQS and WQS are sets of two element tuples each. Equation 3.6 on the facing page states that all q_i elements in the RQS are subsets of set of replicas of the QP Q. Equation 3.7 also states that all $sq_{i,j}$ are subsets of the set of replicas of the QP Q. The ... in Equation 3.5 and Equation 3.15 indicate that there can be many tuple of quorum and subsets of replicas of that quorum. The next restriction posed on the elements of the RQS is shown in Equation 3.8. It states that q_i is a RQ of QP Q. Equation 3.9 states that for all tuples $(q_i, sq_{i,m})$ $sq_{i,m}$ must be supersets of q_i. This is so, when evaluating the cost of an operation, the minimal quorum is used such that we evaluate the minimal average costs. No quorum must be represented more than once in the RQS, this is stated in Equation 3.10. Otherwise, the RQS would not be a set and it would be impossible to evaluate the $a_r(p)$ and $c_r(p)$ expressions. Equation 3.11 stats that no superset of any quorum must be part of any other set of supersets of any other quorum, such that no set of replicas is represented twice in the RQS. Finally, Equation 3.12 requires that all $sq_{i,j}$ are supersets of the q_i that consists of the fewest number of replicas. This guarantees that the smallest quorum is chosen for execution of the operation.

The read operation availability $a_r(p)$ is then calculated as shown in Equation 3.13 on the preceding page. The term $p^{|q|}(1-p)^{N-|q|}$ gives the probability that a set of replicas $|q|$, a RQ, out of the set of all replicas N is available when each replicas has the given p-value. The same is true for the term $p^{|t|}(1-p)^{N-|t|}$, the only difference being that t is a superset to a quorum. All these availabilities are totaled, the result is the $a_r(p)$ for the given QP for the given p-value.

The read costs $c_r(p)$ is then calculated as shown in Equation 3.13. For the calculation of $c_r(p)$ it is necessary to divide by $a_r(p)$ as otherwise the resulting costs would be scaled by the availability mass. The probabilities that q is availability is multiplied by the number of replicas the quorum consists of. The probabilities that t is availability is multiplied by the number of replicas the q consists of. This is because QPs will only use the minimum number of replicas required to create a RQ. In Section 4.1 it will be shown why this is an important requirement.

WQSs differs from RQSs in that the elements q_i must be a WQs instead of a RQs, as shown in Equation 3.18 on the facing page. The calculations for $a_w(p)$ and $c_w(p)$ are otherwise equal to the calculations for $a_r(p)$ and $c_r(p)$.

3.4.1 Example

To better illustrate the construction and use of the RQS and WQS we use the previously used QP as an example. Let Q be a QP that is using 5 replicas. The replicas are named $0, 1, 2, 3$, and 4. A RQ has to read the majority of the replicas and a WQ also has to write the majority of the replicas.

This is enough information to construct the RQS and the WQS for this QP. The RQS is shown in Table 3.2 on the next page and the WQS is shown in Table 3.3 on the following page.

RQS ={
({0,1,2},{{0,1,2,3},{0,1,2,4},{0,1,2,3,4}})
({0,1,3},{{0,1,3,4}})
({0,1,4},{})
({0,2,3},{{0,2,3,4}})
({0,2,4},{})
({0,3,4},{})
({1,2,3},{{1,2,3,4}})
({1,2,4},{})
({1,3,4},{})
({2,3,4},{})
}

Table 3.2: RQS of example the QP.

WQS ={
({0,1,2},{{0,1,2,3},{0,1,2,4},{0,1,2,3,4}})
({0,1,3},{{0,1,3,4}})
({0,1,4},{})
({0,2,3},{{0,2,3,4}})
({0,2,4},{})
({0,3,4},{})
({1,2,3},{{1,2,3,4}})
({1,2,4},{})
({1,3,4},{})
({2,3,4},{})
}

Table 3.3: WQS of example the QP.

The RQS shown in Table 3.2 and the WQS shown in Table 3.3 are identical as both read and write operation require a majority of the replicas to function.

Inserting the RQS and the WQS in the equations for $a_r(p)$ and $a_w(p)$ yields the intermediate step as shown in Table 3.4 and Table 3.5. Inserting the RQS and the WQS in equations for $c_r(p)$ and $c_w(p)$ yields the intermediate step as shown in Table 3.6 on the next page and Table 3.7 on the following page.

$$
\begin{aligned}
a_r(p) =& p^3(1-p)^{5-3} + p^4(1-p)^{5-4} + p^4(1-p)^{5-4} + p^5(1-p)^{5-5} \\
&+ p^3(1-p)^{5-3} + p^4(1-p)^{5-4} \\
&+ p^3(1-p)^{5-3} \\
&+ p^3(1-p)^{5-3} + p^4(1-p)^{5-4} \\
&+ p^3(1-p)^{5-3} \\
&+ p^3(1-p)^{5-3} \\
&+ p^3(1-p)^{5-3} + p^4(1-p)^{5-4} \\
&+ p^3(1-p)^{5-3} \\
&+ p^3(1-p)^{5-3} \\
&+ p^3(1-p)^{5-3}
\end{aligned}
$$

Table 3.4: Intermediate step of preparing the RQS for the read availability $a_r(p)$.

$$
\begin{aligned}
a_w(p) =& p^3(1-p)^{5-3} + p^4(1-p)^{5-4} + p^4(1-p)^{5-4} + p^5(1-p)^{5-5} \\
&+ p^3(1-p)^{5-3} + p^4(1-p)^{5-4} \\
&+ p^3(1-p)^{5-3} \\
&+ p^3(1-p)^{5-3} + p^4(1-p)^{5-4} \\
&+ p^3(1-p)^{5-3} \\
&+ p^3(1-p)^{5-3} \\
&+ p^3(1-p)^{5-3} + p^4(1-p)^{5-4} \\
&+ p^3(1-p)^{5-3} \\
&+ p^3(1-p)^{5-3} \\
&+ p^3(1-p)^{5-3}
\end{aligned}
$$

Table 3.5: Intermediate step of preparing the WQS for the read availability $a_w(p)$.

$$
\begin{aligned}
c_r(p) =& (3(p^3(1-p)^{5-3}) + 3(p^4(1-p)^{5-4}) + 3(p^4(1-p)^{5-4}) + 3(p^5(1-p)^{5-5}) \\
&+3(p^3(1-p)^{5-3}) + 3(p^4(1-p)^{5-4}) \\
&+3(p^3(1-p)^{5-3}) \\
&+3(p^3(1-p)^{5-3}) + 3(p^4(1-p)^{5-4}) \\
&+3(p^3(1-p)^{5-3}) \\
&+3(p^3(1-p)^{5-3}) \\
&+3(p^3(1-p)^{5-3}) + 3(p^4(1-p)^{5-4}) \\
&+3(p^3(1-p)^{5-3}) \\
&+3(p^3(1-p)^{5-3}) \\
&+3(p^3(1-p)^{5-3}))/a_r(p)
\end{aligned}
$$

Table 3.6: Intermediate step of preparing the RQS for the read availability $c_r(p)$.

$$
\begin{aligned}
c_w(p) =& (3(p^3(1-p)^{5-3}) + 3(p^4(1-p)^{5-4}) + 3(p^4(1-p)^{5-4}) + 3(p^5(1-p)^{5-5}) \\
&+3(p^3(1-p)^{5-3}) + 3(p^4(1-p)^{5-4}) \\
&+3(p^3(1-p)^{5-3}) \\
&+3(p^3(1-p)^{5-3}) + 3(p^4(1-p)^{5-4}) \\
&+3(p^3(1-p)^{5-3}) \\
&+3(p^3(1-p)^{5-3}) \\
&+3(p^3(1-p)^{5-3}) + 3(p^4(1-p)^{5-4}) \\
&+3(p^3(1-p)^{5-3}) \\
&+3(p^3(1-p)^{5-3}) \\
&+3(p^3(1-p)^{5-3}))/a_w(p)
\end{aligned}
$$

Table 3.7: Intermediate step of preparing the WQS for the read availability $c_w(p)$.

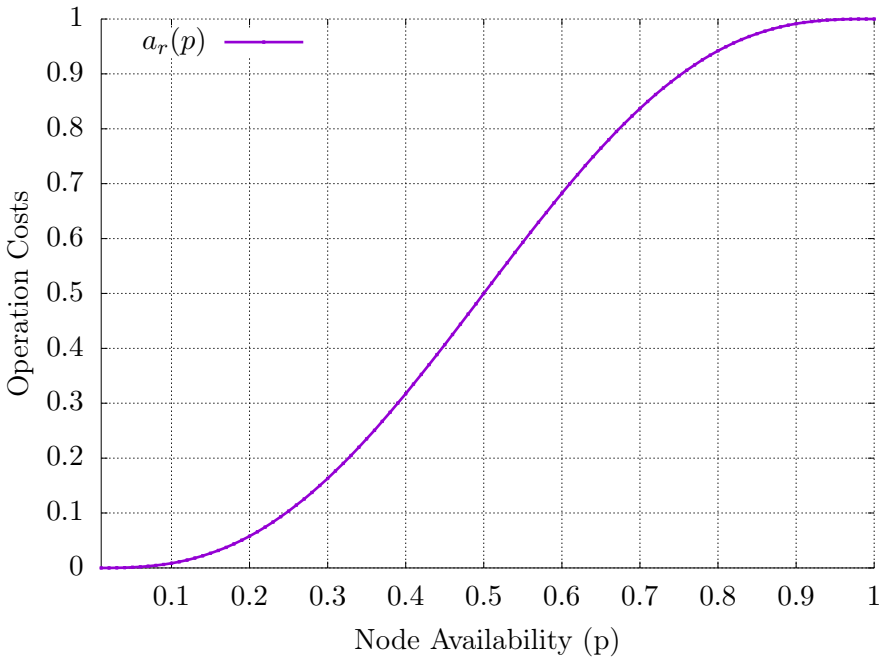


Figure 3.8: The $a_r(p)$ for the given example with p-values iterated from 0.0 to 1.0 in 0.01 steps.

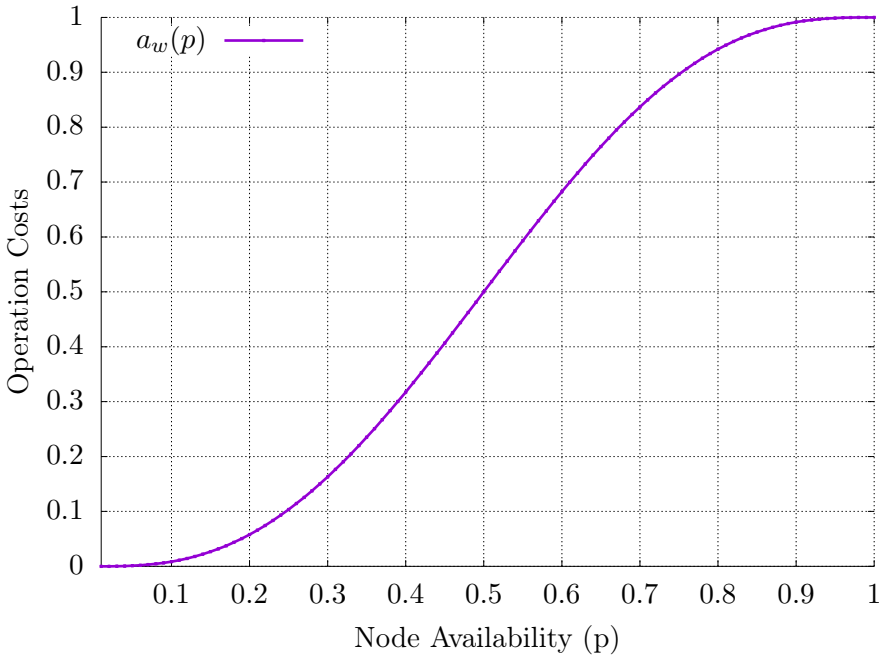


Figure 3.9: The $a_w(p)$ for the given example with p-values iterated from 0.0 to 1.0 in 0.01 steps.

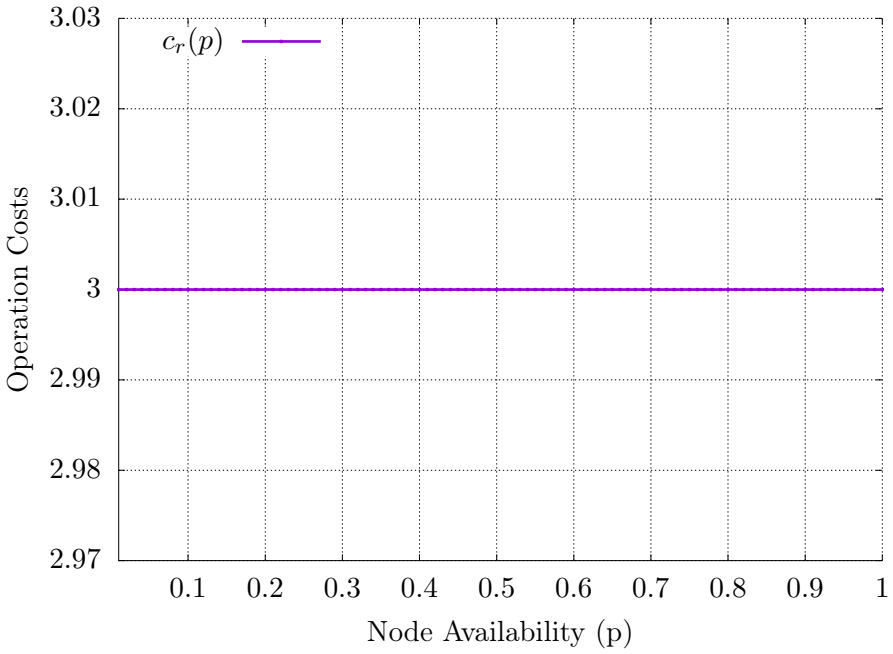


Figure 3.10: The $c_r(p)$ for the given example with p-values iterated from 0.0 to 1.0 in 0.01 steps.

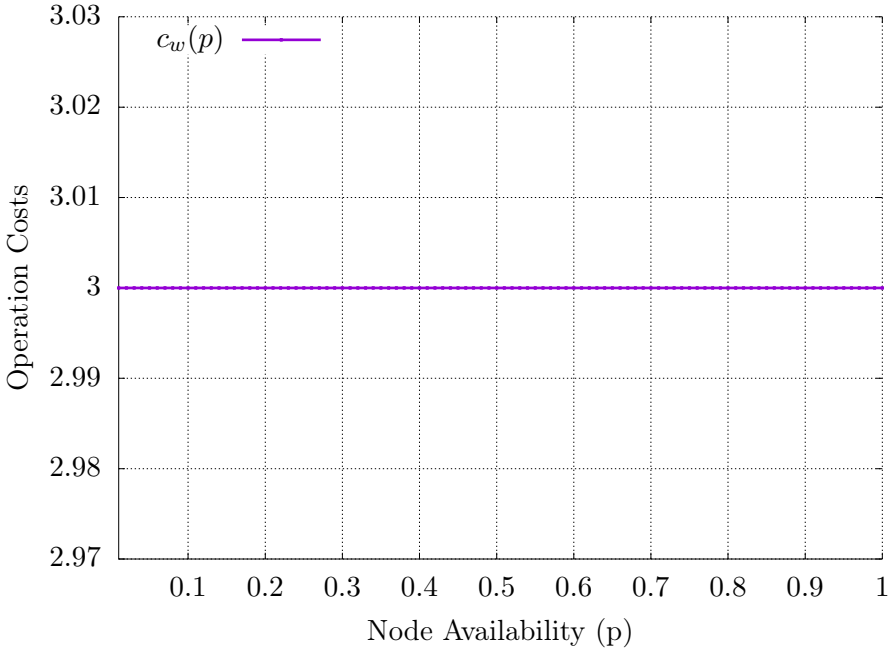


Figure 3.11: The $c_w(p)$ for the given example with p-values iterated from 0.0 to 1.0 in 0.01 steps.

Figure 3.8 on page 21 shows the $a_r(p)$ and Figure 3.9 on page 21 shows the $a_w(p)$ of the example with $p \in [0,1]$. In both plots the x axis represents the p value of all replicas, the y axis shows the availability of the read or write operation. The $c_r(p)$ of the example is shown in Figure 3.10 on the preceding page and $c_w(p)$ of the example is shown in Figure 3.11 on the facing page. The cost for both operation is 3 independent of the replica availability. Considering the definition of the QP used in this example, this result is obvious, as the defined QP should use 3 replicas, the majority of the replicas, for reading and writing. Both plots, and all consecutive plots in this work, increment the p values in 0.01 steps between 0 and 1.

This approach to calculate the $a_r(p)$ the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ is used in all following chapters of this work unless stated otherwise.

4 Existing Quorum Protocols

QPs provide highly available access to data by means of replication and at the same time maintain the 1SR property. In this work only pessimistic syntactic DRS QPs are considered. Pessimistic syntactic DRS QPs can be further classified into unstructured quorum protocols (UQPs) and structured quorum protocols (SQPs). SQPs arrange the individual replicas in a GS, and use this GS to formulate rules how to construct quorums. UQPs use no such GS. In order for read and write operations to work, the data of the operations needs to be communicated to and from the replicas of the quorum used. Implicitly, most UQP and SQP rely on a completely connected GS as a communication medium between its replicas. The GS used for communication is called the logical network topology (LNT) in this work. A PNT is a GS that actually arranges and connects the replicas in the real-world. This is because usually, QPs are used on already existing network topologies.

In this chapter three existing QPs are presented. Many more QPs exist, but the presented three are interesting for multiple reasons. The MCS was one of the first QPs, and lends itself as an easily understood introduction into QPs [11]. The GP is a SQP that uses a completely connected GS as a communication network [12]. Finally, the Triangular Lattice Protocol (TLP) is a SQP where the LNT is also used as the PNT [4, 13]. These QPs will later be used as a benchmark to the newly developed QPs and as QPs that will be used in the analysis of the mapping approach that makes it possible to analyze QPs on PNTs.

4.1 Majority Consensus Protocol (MCS)

The MCS is an UQP. The MCS reads a RQ of $\lceil N/2 \rceil$ replicas and writes a WQ of $\lceil (N+1)/2 \rceil$ replicas, where N is the total number of replicas [11]. For a given set of replicas $\mathcal{N}$, QR and QW both are subsets of $\mathfrak{P}(\mathcal{N})$ for which the following

$$\forall qr \in QR, qw \in QW : |qr| + |qw| = N + 1 \tag{4.1}$$

$$\forall qw_1, qw_2 \in QW : |qw_1| + |qw_2| \geq N + 1 \tag{4.2}$$

is true. The MCS uses the intersection of RQs and WQs, and of WQ to maintain 1SR in the previously described manner.

The read availability $a_r(p)$ of the MCS is

$$a_r(p) = \sum_{k=\lceil N/2 \rceil}^{N} \binom{N}{k} p^k (1-p)^{N-k} \tag{4.3}$$

and the write availability $a_w(p)$ is

$$a_w(p) = \sum_{k=\lceil (N+1)/2 \rceil}^{N} \binom{N}{k} p^k (1-p)^{N-k}. \tag{4.4}$$

The read costs $c_r(p)$ are $\lceil N/2 \rceil$. The write costs $c_w(p)$ are $\lceil (N+1)/2 \rceil$ [11, 14]

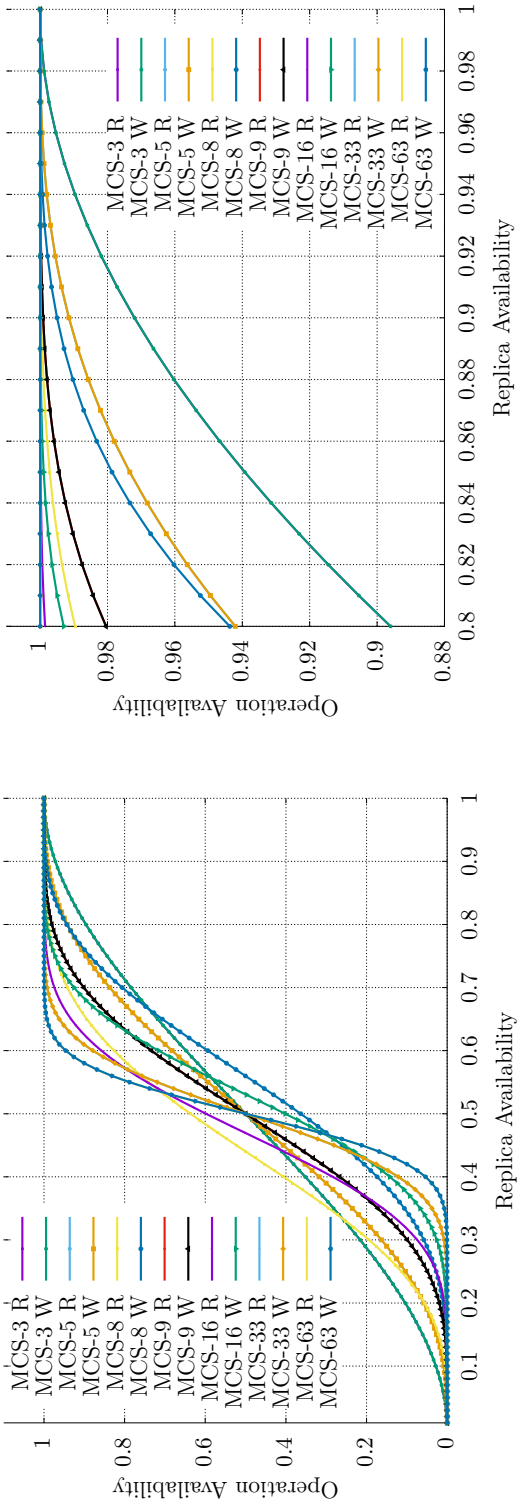
Algorithm 1 and Algorithm 2 give the specific implementations of the procedures *isReadQuorum*, and *isWriteQuorum* for the MCS, so the MCS can be used in conjunction with the RQS and the WQS.

Algorithm 1: Procedure *isReadQuorum*(*replicas*) of the MCS
Input: *replicas* = a set of replicas that is to be tested whether or not it is a RQ for the given MCS
Result: true if $\lvert replicas \rvert \geq \lceil N/2 \rceil$, false otherwise

Algorithm 2: Procedure isWriteQuorum(*replicas*) of the MCS
Input: *replicas* = a set of replicas that is to be tested whether or not it is a WQ for the given MCS
Result: true if $\lvert replicas \rvert \geq \lceil (N+1)/2 \rceil$, false otherwise

Figures 4.1a and 4.1b on the next page show the read and write availability of the MCS with different numbers of replicas. Note that the read and write availabilities are the same for the MCS with an odd number of replicas and therefore the lines in the Figures overlap. In these figures it can be seen that the more replicas the MCS uses the higher the availability is when $p \geq 0.5$. This is especially visible when the MCS uses 63 replicas [1]. In the close-up figure, the difference between the MCS with 3 and 63 is particularly obvious. The 63 replicas variant has an availability of basically 1 and the 3 replicas variant starts out at a availability of < 0.9. The drawback of the MCS is that costs per operation increase linearly with the number of replicas used by the MCS. This can be seen in Figure 4.2 on page 28.

[1] Indicated by label MCS-63 in the Figures



(a) The read (R) and write (W) operation availability of the MCS with different numbers of replicas.

(b) The read (R) and write (W) operation availability of the MCS with different numbers of replicas with $p \geq 0.8$.

Figure 4.1: MCS read and write operation availability.

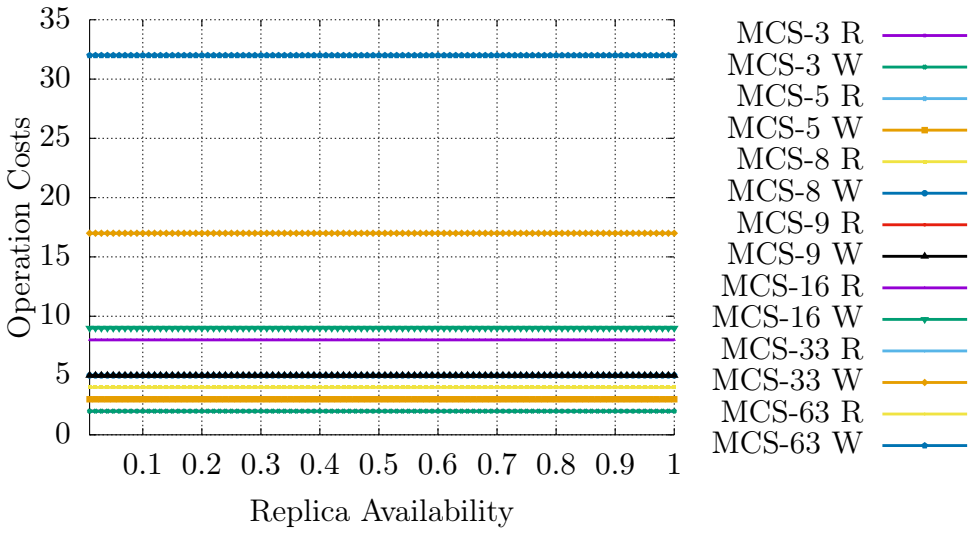


Figure 4.2: The read (R) and write (W) operation cost of the MCS with different numbers of replicas.

4.2 Grid Protocol (GP)

The GP is a SQP. The GP was presented by Kumar in [12]. The read and write operation of the QP is proven to have the 1SR property. A grid used by the GP is shown in Figure 4.3. RQs are constructed by selecting one replicas of each column. For a grid with R rows and C column the read availability is:

$$a_r(p) = (1-(1-p)^C)^R \tag{4.5}$$

[14]. The read costs for the GP is equal to the number of columns of the grid used. As an example for a valid RQ for the GP in Figure 4.4 on the following page consider the replicas $\{0,5,6,15\}$. WQs are constructed by selecting one replica of each column and one complete column. A example of a WQ is given in Figure 4.5

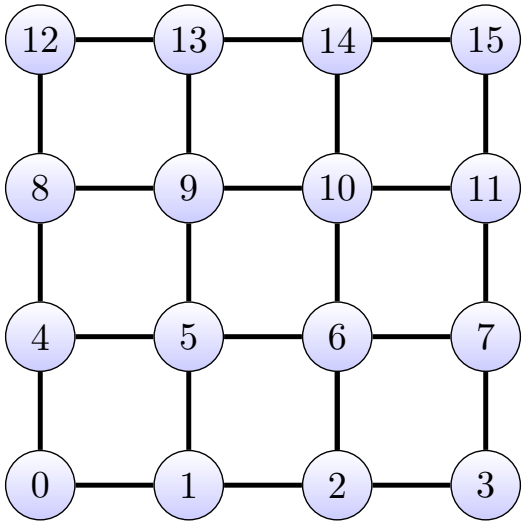


Figure 4.3: GS used by the Grid Protocol.

on the next page. The WQ consists of the replicas $\{2,3,4,6,9,10,14\}$. The replicas of the RQ and the replicas of the WQ intersect in replica 6. The write availability of the GP is:

$$a_w(p) = (1-(1-p)^C)^R - (1-p^C-(1-p)^C)^R \tag{4.6}$$

[14]. The write costs for the GP is equal to $C+R-1$ [15]. Figure 4.6a on page 31 gives examples of the read and write availability of the GP with 4 to 64 replicas. The grids used in this example are all square. It can be seen that with more replicas the read availability increases significantly. The write availability on the other hand increases slowly until a high p value is reached, this can be seen in Figure 4.6b on page 31. Figure 4.7a on page 33 shows examples of the read and write availability, of the QP, when the grid used is not square. For the 8×1 grid the read availability is very good as there is a very high chance of finding one

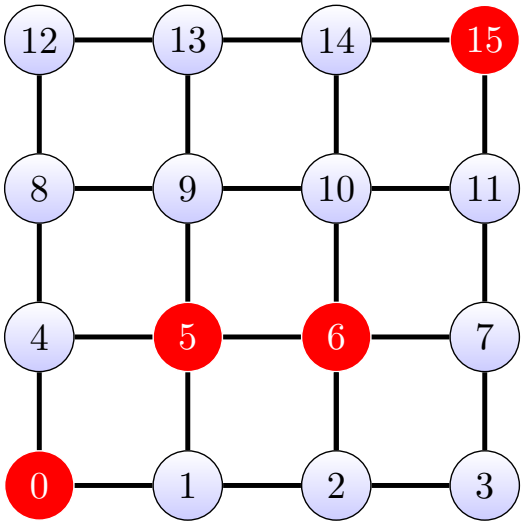


Figure 4.4: An example of a RQ used by the Grid Protocol. The red colored replicas are the elements of the RQ.

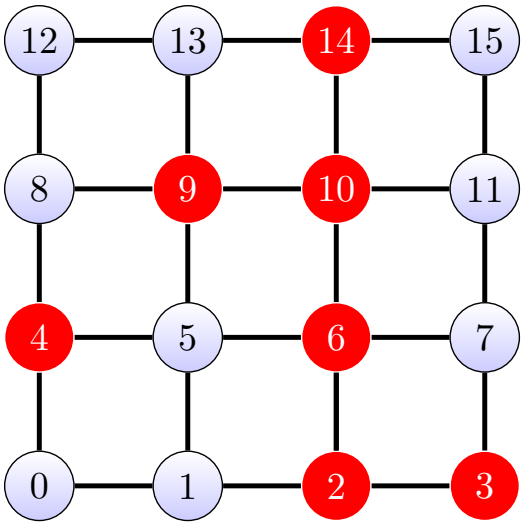
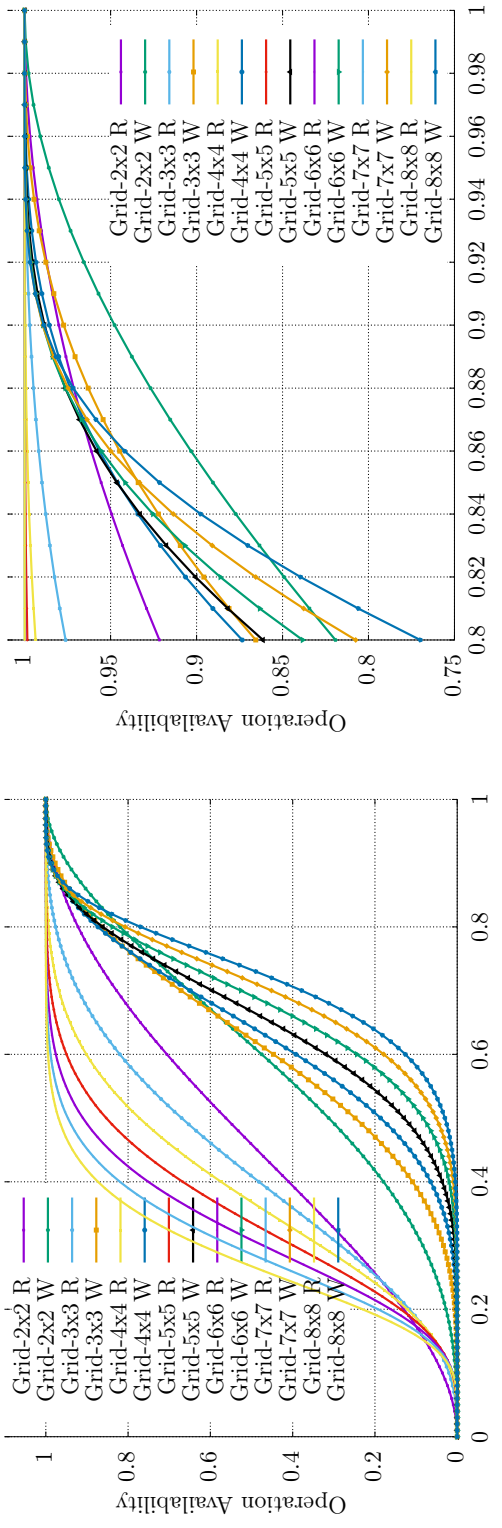


Figure 4.5: An example of a WQ used by the Grid Protocol. The red colored replicas are the elements of the WQ.



(a) The Read (R) and Write (W) operation availability of the GP with different number of replicas, but the same number of replicas per column and row.

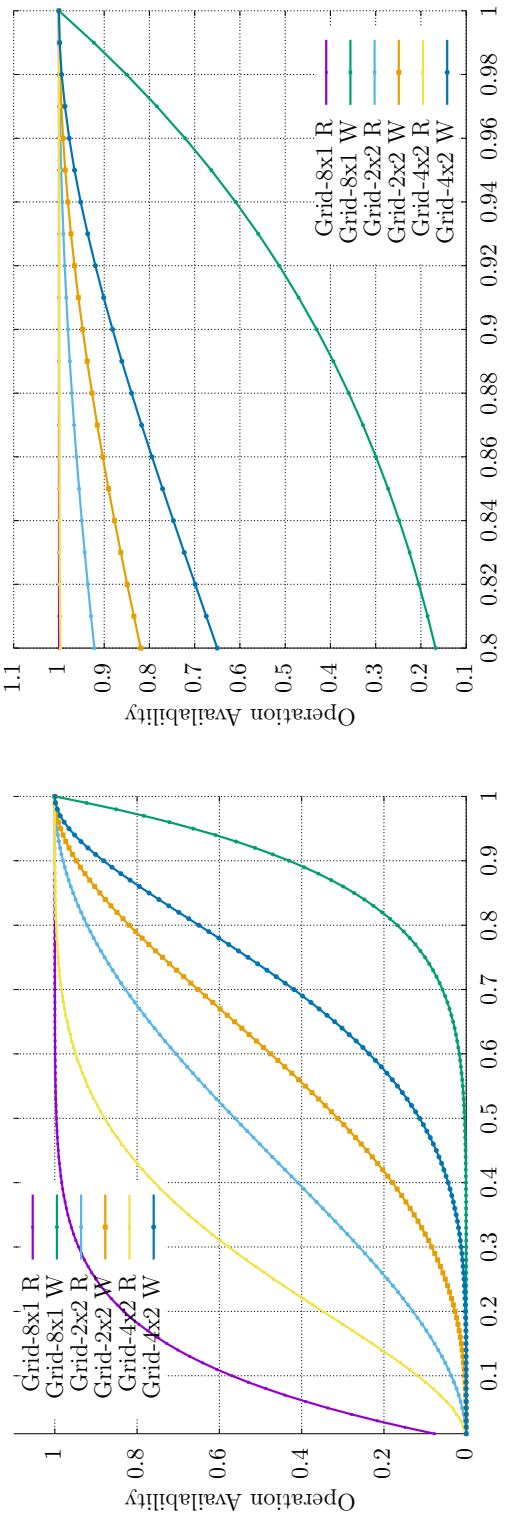
(b) The Read (R) and Write (W) operation availability of the GP with different number of replicas, but the same number of replicas per column and row, with $p \geq 0.8$.

Figure 4.6: GP read and write operation availability for symmetrical GS.

available replica in eight. The write availability on the other hand is very poor. In the example the write availability of the 8×1, and 4×2 GP exceeds the write availability of the nonreplicated case only when $p \geq 0.92$. This is because finding one complete column is very unlikely in these cases as shown in Figure 4.7b on the next page. Therefore, using the GP in such a configuration only would make sense if writing data would be the exception.

Even though there are formulas to compute $a_r(p)$, $a_w(p)$, $c_r(p)$, and $c_w(p)$, for a later step an algorithm is needed to construct the RQS and the WQS for a GP. Algorithm 3 on page 34 and Algorithm 4 on page 34 give the specific implementations of the procedures *isReadQuorum*, and *isWriteQuorum* for the GP. In such a way that the RQS and WQS can be constructed for the GP. The procedure *isReadQuorum* for the QP simply tests if there is an available replica in each column of the grid. If the procedure finds a column where this is not the case, then false is returned. If all columns have available replicas, true is returned. The procedure *isWriteQuorum* for the QP returns true, if the available replicas *replicas*[2] are a superset of a column and if the replicas are also sufficient to constitute a RQ. Otherwise, false is returned.

[2] *replica* being the parameter passed to the *isWriteQuorum* and *isReadQuorum* procedure



(a) The Read (R) and Write (W) operation availability of the GP with different number of replicas, and different number of replicas per column and row.

(b) The Read (R) and Write (W) operation availability of the GP with different number of replicas, and different number of replicas per column and row, $p \geq 0.8$.

Figure 4.7: GP read and write operation availability for asymmetrical GS.

Algorithm 3: Procedure *isReadQuorum*(*replicas*) of the GP

```
Input: replicas = a set of replicas that is to be tested whether or not it is
       a RQ for the given GP
Data: C = the set of sets of replicas defining the columns of the grid
Result: true if replicas contains a replica of each column
1 for c ∈ C do
2 |   if c ∩ replicas = ∅ then
3 |   |   return false
4 |   end
5 end
6 return true
```

Algorithm 4: Procedure *isWriteQuorum*(*replicas*) of the GP

```
Input: replicas = a set of available replicas that is to be tested whether
       or not it is a RQ for the given GP
Data: C = the set of sets of replicas defining the columns of the grid
Result: true if replicas contains a replica of each column and replicas
        contains all replicas of one column
1 for c ∈ C do
2 |   if replicas ⊇ c ∧ isReadQuorum(replicas) then
3 |   |   return true
4 |   end
5 end
6 return false
```

4.3 Triangular Lattice Protocol (TLP)

The TLP is a SQP based on the GP. It provides 1SR consistency as proven in [13]. The grid arrangement of the replicas of the GP is extended to a triangulated grid. Figure 4.8 shows an example of a GS used by the TLP. Every RQ of the TLP consists of a complete vertical or a horizontal path through the GS. Every WQ of the TLP consists of a complete vertical and a complete horizontal path through the GS. An example of a horizontal path is $(4,5,9,10,11)$ as shown in Figure 4.9 on the following page. Figure 4.10 on the next page shows an example for a vertical path consisting of the replicas $(1,5,6,10,14)$. In Figure 4.11, the diagonal path

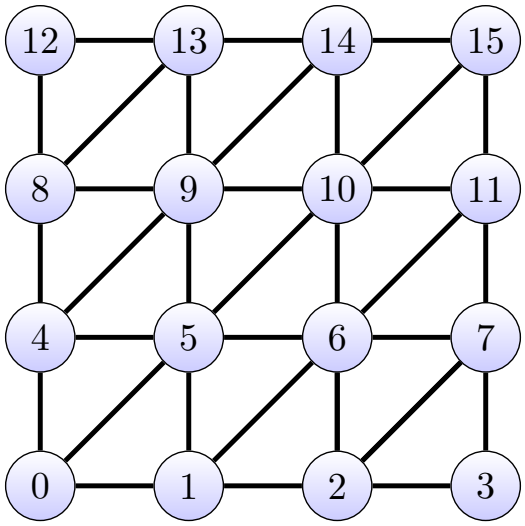


Figure 4.8: A GS used by the Triangular Lattice Protocol.

$(0,5,10,15)$ connecting the replicas creates a minimal path that crosses the GS vertically and horizontally. The minimal path is a connected replicas that crosses a GS vertically and horizontally.

This diagonal path can be used as a very efficient WQ. Given the three example it can be seen how read and WQs intersect.

As quorums for the TLP are created by finding paths through a GS, no simple closed formula exists that calculates the read and write availability or costs. Therefore, the RQS, and WQS are used in combination with Algorithm 7 on page 38 and Algorithm 8 on page 38 to calculate the read and write availability, and the operation cost for the TLP [13]. The procedure isWriteQuorum tests if a set of replicas is a vertical and a horizontal path in a given GS. These to checked by the two procedures isVerticalPath shown in Algorithm 5 on page 37 and isHorizontalPath shown in Algorithm 6 on page 38. Both procedures follow the same idea, in that they iterate two sets of replicas and see if there is a path between them.

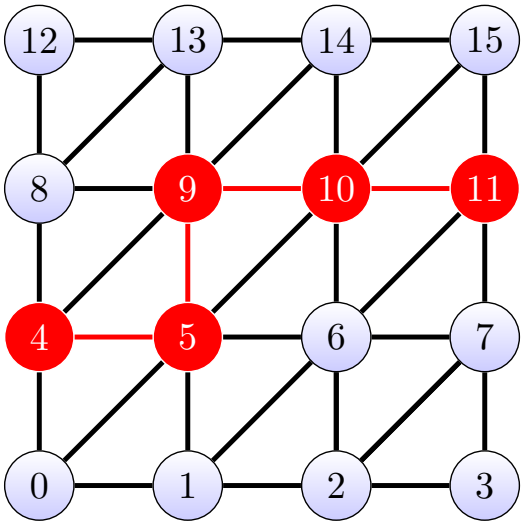


Figure 4.9: A horizontal path through the GS used by the TLP.

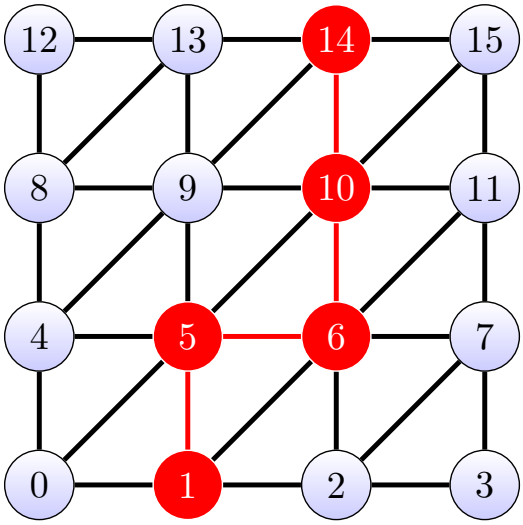


Figure 4.10: A vertical path through the GS used by the TLP.

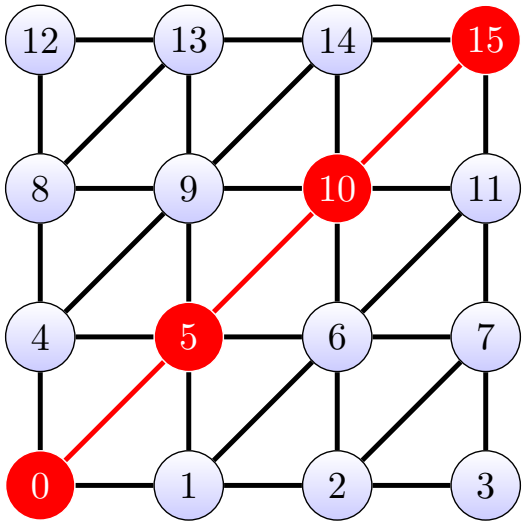


Figure 4.11: A diagonal path through the GS used by the TLP.

Algorithm 5: Procedure isVerticalPath(*replicas*)

Input: *replicas* = a set of replicas that is to be tested whether or not they form a vertical path
Data : *top* = Set of replicas that are elements of the top row of the GS used by the TLP
bottom = Set of replicas that are elements of the bottom row of the GS used by the TLP
Result: true if *replicas* form a vertical path

```
1 for t ∈ top ∩ replicas do
2 |   for b ∈ bottom ∩ replicas do
3 |   |   if Ω(t,b) ∈ replicas then
4 |   |   |   return true
5 |   |   end
6 |   end
7 end
8 return false
```

Algorithm 6: Procedure isHorizontalPath(*replicas*)

Input: *replicas* = a set of replicas that is to be tested whether or not they form a horizontal path
Data : $left$ = Set of replicas that are elements of the left column of the GS used by the TLP
right = Set of replicas that are elements of the right column of the GS used by the TLP
Result: true if *replicas* form a horizontal path

```
1 for t ∈ left ∩ replicas do
2 |   for b ∈ right ∩ replicas do
3 |   |   if Ω(t,b) ∈ replicas then
4 |   |   |   return true
5 |   |   end
6 |   end
7 end
8 return false
```

Algorithm 7: Procedure isReadQuorum(*replicas*) of the TLP

Input: *replicas* = a set of replicas that is to be tested whether or not it is a RQ for the given TLP
Result: true if *replicas* form a RQ, false otherwise

```
1 return isHorizontalPath(replicas) ∨ isVerticalPath(replicas)
```

Algorithm 8: Procedure isWriteQuorum(*replicas*) of the TLP

Input: *replicas* = a set of replicas that is to be tested whether or not it is a WQ for the given TLP
Result: true if *replicas* form a WQ, false otherwise

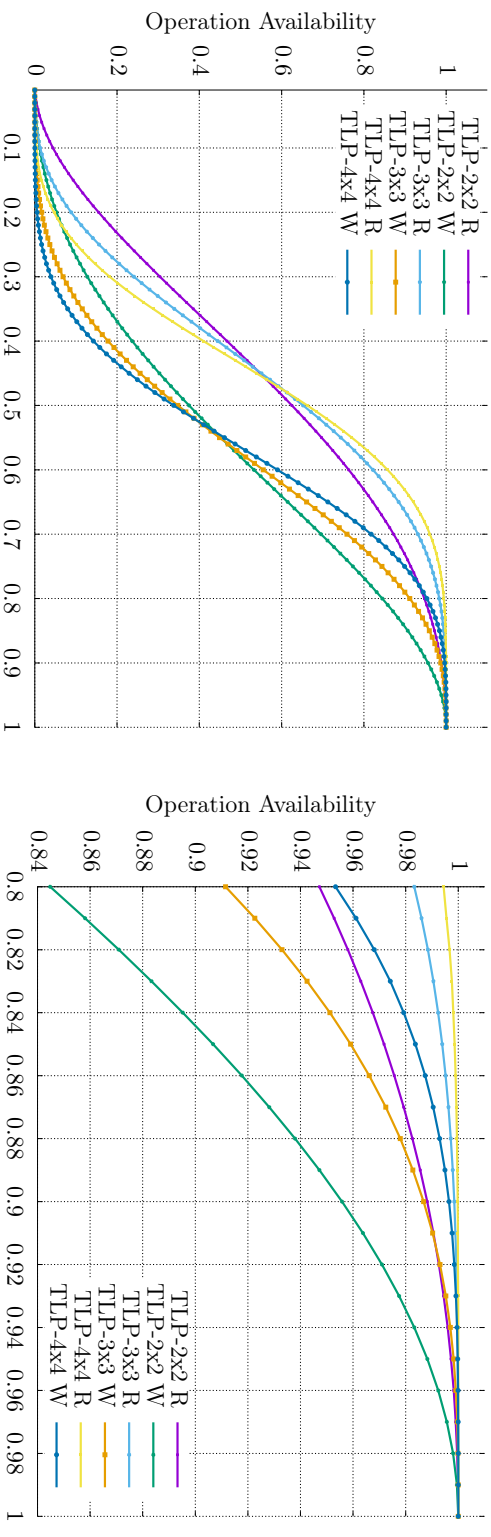
```
1 return isHorizontalPath(replicas) ∧ isVerticalPath(replicas)
```

Figure 4.12a on the following page shows the $a_r(p)$ and the $a_w(p)$ of the TLP on various $x \times x$ triangular lattice GSs. It can be seen that the TLP on the 4×4 GS has a higher $a_r(p)$ the other TLPs as soon as $p > 0.5$. The write availability $a_w(p)$ of the TLP on the 4×4 GS becomes the highest write availability as soon as $p > 0.57$. Figure 4.12b on the next page shows a close-up of Figure 4.12a where $p \geq 0.8$. Here again, it can be seen that the TLP on a 4×4 GS has the highest $a_r(p)$ and $a_w(p)$. Figure 4.13 on page 41 shows the $c_r(p)$ and $c_w(p)$ of the four tests $x \times x$ TLP variants. Here it can be seen that if the GS has fewer vertices the costs per operation decreases. Various $x \times y$ GSs were tested with the TLP. The $a_r(p)$ and the $a_w(p)$ of these GSs are shown in Figure 4.14a on page 42. Especially the 8×1 GS is notable. Its $a_r(p)$ is extremely good as the minimal RQ consists of only a single replica. The minimal WQ on the other hand, requires all eight replicas. This can also be seen in the cost analysis shown in Figure 4.14b on page 42.

Figure 4.15a on page 43, Figure 4.15b, and Figure 4.16 show the comparison of the GP and the TLP on a 5×5 GS. Here the $a_r(p)$ of the GP is higher than the $a_r(p)$ of the TLP. The $a_w(p)$ of the TLP is higher than the $a_w(p)$ of the GP. The costs analyses shows little differences between the $c_r(p)$ of both protocols. The $c_w(p)$ of the TLP is a lot cheaper than the $c_w(p)$ of the GP.

The increased write costs of the TLP with p not close to 1 nor to 0 has the following explanation. With very low p values it is most likely that the only vertical and horizontal path available is the diagonal, as this path consists of the least amount of replicas. With very high p values many vertical and horizontal paths are available, including the diagonal path, as the minimal vertical and horizontal path is chosen by TLP the diagonal is selected again. In between more WQs are available, but these consist of more replicas than the diagonal, explaining the higher costs. This effect is not as pronounced for the read operation of the TLP.

Concluding, it can be said that the TLP provides a write operation that has a higher availability than the write operation of the GP. For $p \geq 0.8$ the read operation availability of the two protocols are nearly identical, especially for GSs consisting of more than 16 replicas.



(a) The read (R) and write (W) operation availability of the TLP with different number of replicas, but the same number of replicas per column and row.

(b) The read (R) and write (W) operation availability of the TLP with different number of replicas, but the same number of replicas per column and row, with $p \geq 0.8$.

Figure 4.12: TLP read and write operation availability for symmetrical GS.

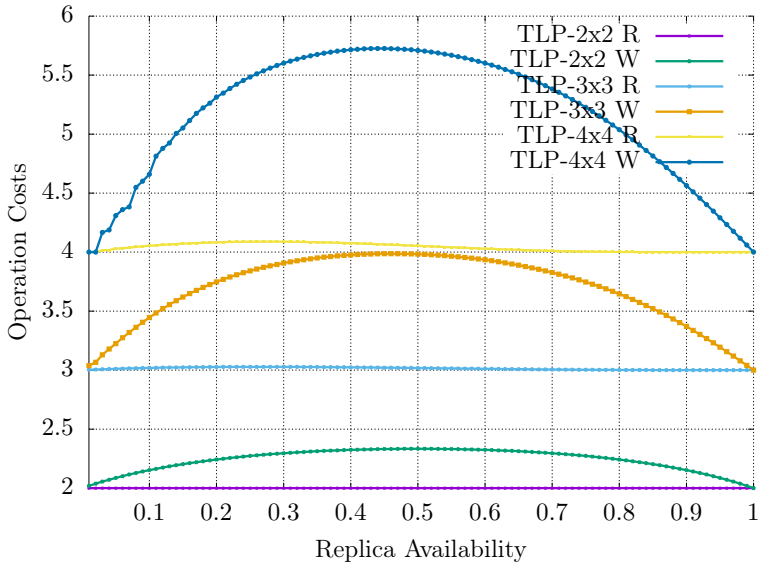
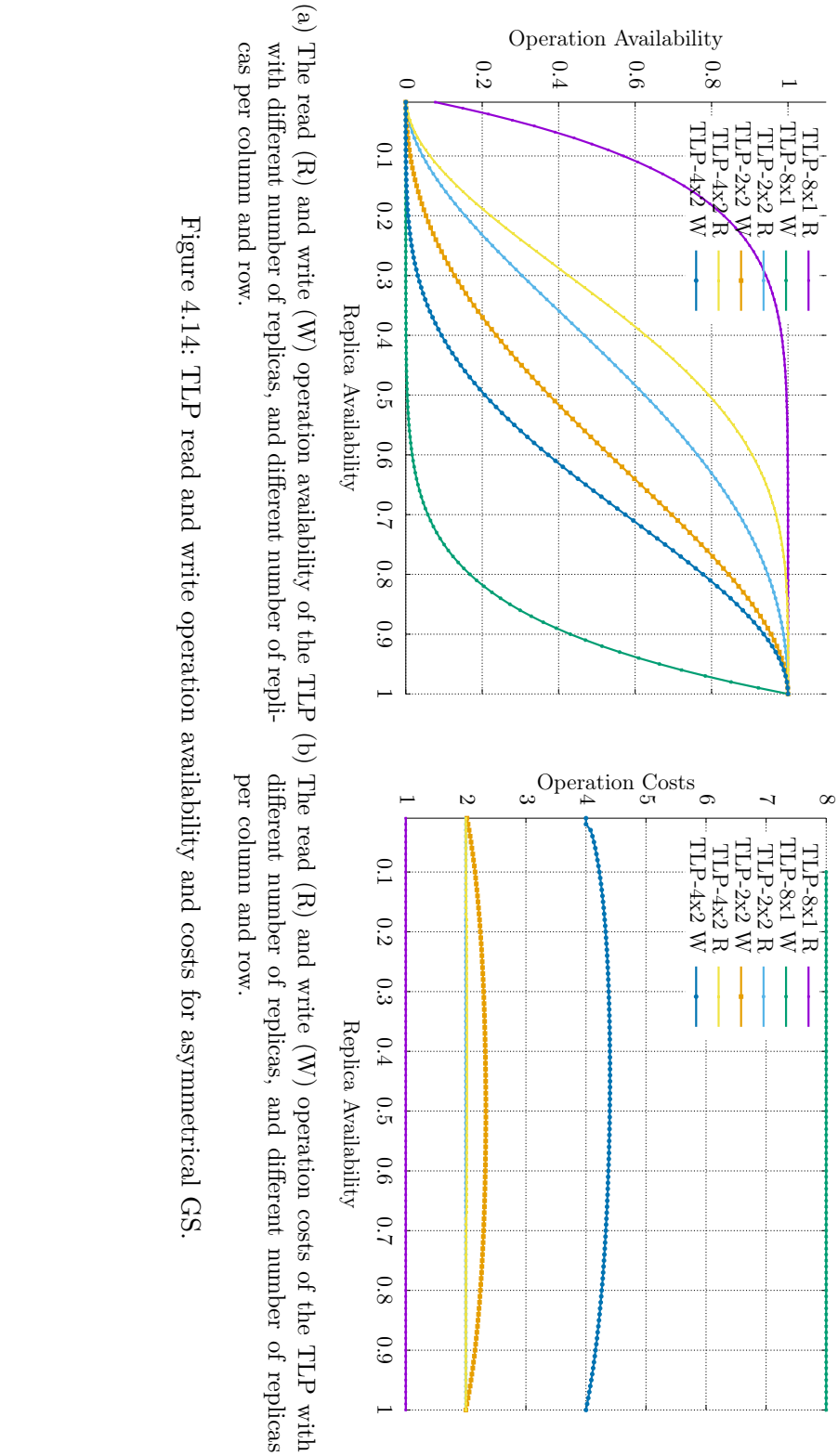


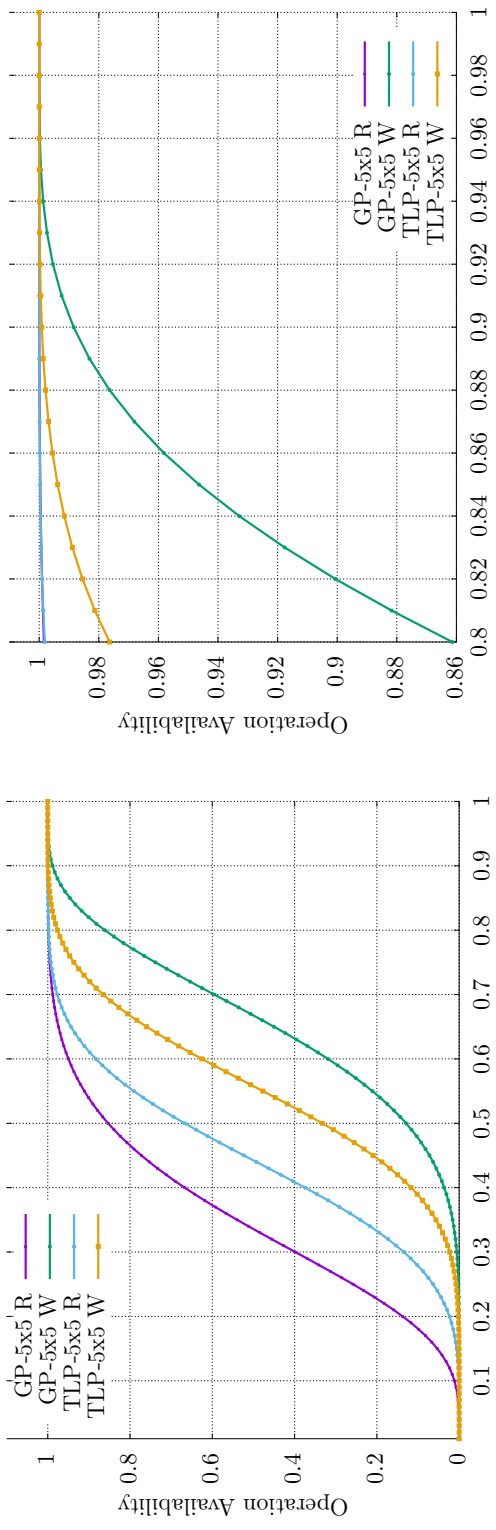
Figure 4.13: The read (R) and write (W) operation costs of the TLP with different number of replicas, but the same number of replicas per column and row.



(a) The read (R) and write (W) operation availability of the TLP with different number of replicas, and different number of replicas per column and row.

(b) The read (R) and write (W) operation costs of the TLP with different number of replicas, and different number of replicas per column and row.

Figure 4.14: TLP read and write operation availability and costs for asymmetrical GS.



(a) The read (R) and write (W) availability of GP 5×5 versus TLP 5×5.

(b) The read (R) and write (W) availability of GP 5×5 versus TLP 5×5 with $p \geq 0.8$.

Figure 4.15: GP and TLP read and write operation availability.

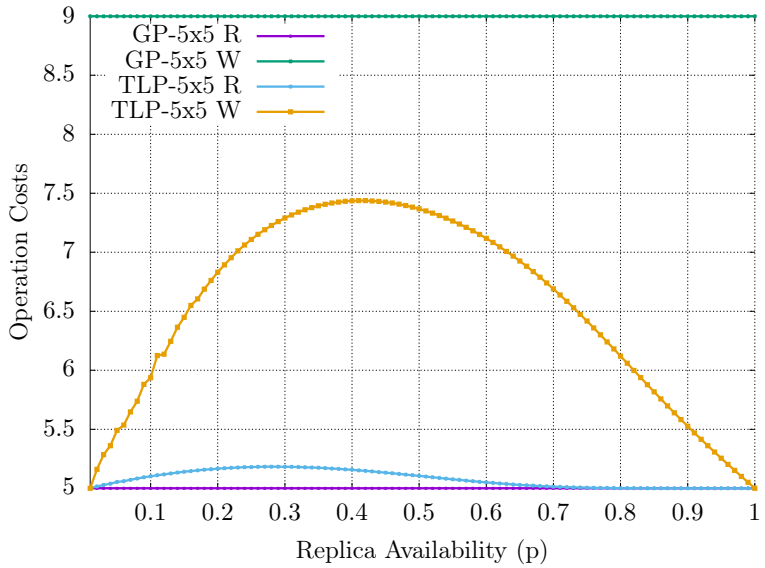


Figure 4.16: The read (R) and write (W) costs of GP 5×5 versus TLP 5×5.

5 Mappings of Quorum Protocols to Physical Network Topologies

This chapter is based on the work published, by the author, in [rsot17long, 16].

5.1 Logical Network Topologies vs. Physical Network Topologies

Previously, it was explained that QPs use replicas to create RQ and WQs that execute read and write operations. It was not explained how the data required to execute these operations is communicated to the replicas of these quorums. This section will show that the GS used to communicate the data has to be considered in the analyses of the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$.

Consider the MCS. The MCS is an UQP where each quorum is a subset of all the replicas used by the QP. It is not specified how the data that is read or written is communicated to or between the replicas. Implicitly, it is assumed that all replicas can directly communicate with each other or the outside world over some media. When someone or something wants the QP to perform an operation on the replicas, this intent has to be communicated to all replicas that should be involved in this operation. In this work, it is assumed that replicas communicate which each other to perform the desired operation. GSs are used to model the communication structure between the replicas. Where the vertices of the GS are the replicas and the edges are the communication links. To replicas $0, 1$ can communicate in an GS G when they are connected as defined in Section 2.3 and all the vertices of the connecting path are available. This implicitly or explicitly assumed communication structure, used by a QP, is called its logical network topology (LNT) in this work. The MCS with five replicas would have a LNT as shown in Figure 1.6 on page 4. In this figure, it can be seen that each replica has a directly link to every other replica of the LNT. It is unlikely that existing communication structures are equal to the LNT required by the QP. The GS that is actually used to communicate the data between the replicas, is called the physical network topology (PNT). For example, consider the PNT in Figure 5.1 on the next page. It can be seen that the GS in Figure 1.6 is not isomorphic to the GS in Figure 5.2 on the following page [17]. This means that some mapping between the LNT and PNT has to take place when communication between replicas during an operation execution is required.

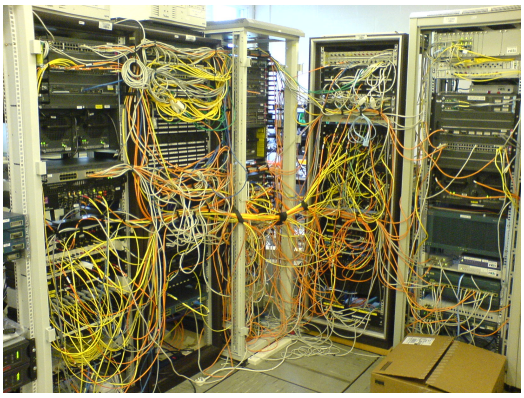


Figure 5.1: A communication structure found in the real world.

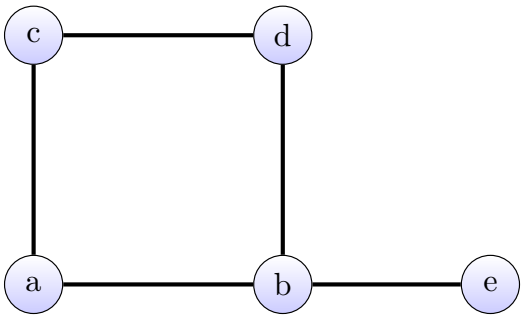


Figure 5.2: GS used by as a PNT.

In the following, it is shown why the PNT has to be considered in the availability and cost analysis of a QP.

Let $s = 0,3,4$ be a RQ used by the MCS. In Figure 5.2 replace 0 is donated a, 1 by b, and so forth. On the GS in Figure 5.2 these three replicas are not connected and can therefore not communicate with each other. This is a problem, as the cost and availability analysis of the MCS assumes that these replicas of the quorum can communicate. But in this example, this assumption does not hold. Only if the replica with ID 1 is added to s, such that $s = 0,1,3,4$, then the replicas of the RQ s can communicate with each other. But this means that this quorum is now less likely to be available, because it now consists of four instead of three replicas, and four replicas are less likely to be simultaneously available than three replicas with $p < 1.0$.

In order to increase the precision of the $a_r(p)$, $a_w(p)$, $c_r(p)$, and the $c_w(p)$ analysis of the QP analyzed, the PNT that is used as a communication network has to be considered. The mapping approach presents one possibility to increase the precision of the analysis, by considering a given PNT as a communication network that is used by during the analysis of the QP and not the LNT assumed by the QP.

No other work could be found that considers the PNT or any similar concept.

5.2 Mapping Definition

A mapping is an injection from one GS, the LNT, to another GS, the PNT. This requires that the number of vertices in the codomain structure is at least equal to the number of the vertices of the GS. Let G and G' be two GSs, then a mapping $M(G,G')$, is an bijection, as defined:

$$
\begin{aligned}
M(G,G') = \{(v_i, v_i')| & && (5.1)\\
v_i \in G_V & && (5.2)\\
\wedge\, v_i' \in G_V' & && (5.3)\\
\wedge\, \forall (v,v'),(v,v'') \in M : v' = v'' & && (5.4)\\
\wedge\, \forall (v',v),(v'',v) \in M : v' = v''\} &
\end{aligned}
$$

As an example, let G be the GS from Figure 1.6, let G' be the GS from Figure 5.2, and let $\{(0,e),(1,d),(2,c),(3,b),(4,a)\} = M(G,G')$ be a mapping. Figure 5.3 on the next page shows this mapping graphically, where the arrows show which vertex is mapped to which vertex in the other GS. Function $\mathfrak{I}$ is uses the mapping $M(G,G')$ to determine the vertex in G' the vertex v of G is mapped to.

$$
\mathfrak{I}(v, M(G,G')) = v' \text{ where } (v,v') \in M(G,G') \tag{5.5}
$$

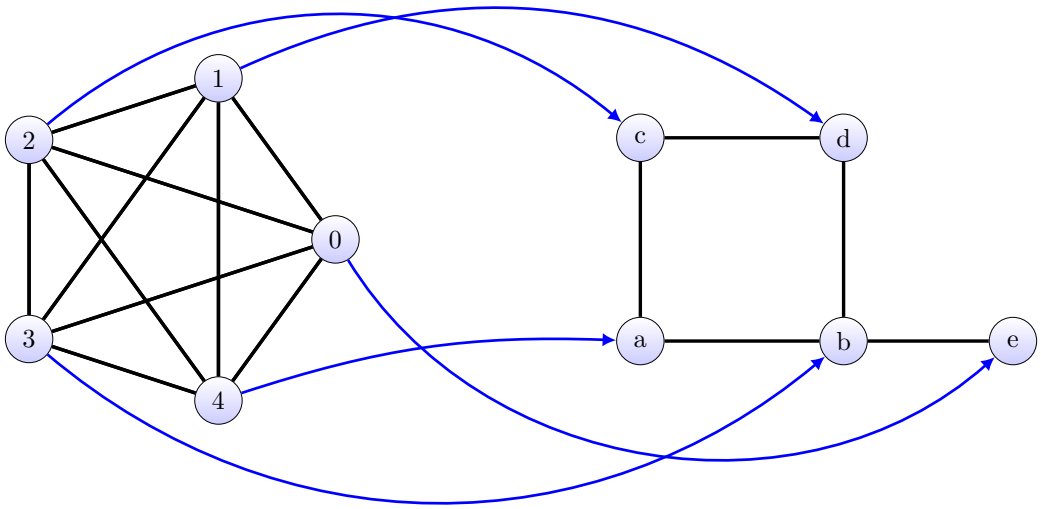


Figure 5.3: A mapping from of a LNT in Figure 1.6 to the GS in Figure 5.2.

For example, given the above shown mapping $\mathfrak{I}(2,\{(0,e),(1,d),(2,c),(3,b),(4,a)\}) = c$. The function $\mathfrak{M}$ maps all replicas of the set q of the vertices in G', where $q = \mathbb{V}(G)$.

$$\mathfrak{M}(q, M(G,G')) = \{\mathfrak{I}(v, M(G,G'))|v \in q\} \tag{5.6}$$

For example, given the above shown mapping $\mathfrak{M}(\{2,4\},\{(0,e),(1,d),(2,c),(3,b),(4,a)\}) = \{c,a\}$. The function $\mathfrak{M}$ maps all replicas of the set q of the vertices in G', where $q = \mathbb{V}(G)$. After the QP has selected all necessary replicas to construct a quorum based on its LNT, it is tested if the selected replicas are connected in the PNT. As mappings are usually not between isomorphic GSs, in the general case, it can be assumed that the replicas of the quorum are not connected directly with each other in the PNT [18]. Consequently, the function $\mathfrak{R}$ adds the minimal number of additional replicas required to reestablish the communication between the replicas of the quorum.

$$\mathfrak{R}(q', \text{PNT}) = \{\text{minimal set of replicas that connects the replicas of } q' \text{ including } q'\} \tag{5.7}$$

The minimal set means the set with the fewest number of replicas.

Assuming the mapping shown in Figure 5.3 and a quorum of $\{0,2,4\}$, the function $\mathfrak{R}$ constructs a new set of replicas that includes these three replicas plus, a certian number of additional replicas such that:

$$\forall(v,v') \in \mathfrak{R}(\{0,3,4\}, \text{GS}) \quad | \quad \Omega(v,v') \in \text{GS} \tag{5.8}$$

is true. In this case the resulting set is $\{a,b,c,e\}$. This new set is then used to execute the operation the original quorum was meant to be used for.

So far only one mapping has been considered, but given a PNT with N vertices there are $N!$ possible mappings.

The question becomes, which of the $N!$ mappings should be used? The best mapping should be used, but in order to determine the best mapping, a comparison criteria has to be established. Which comparison criteria is chosen depends on the use case. Generally, the user can define their own comparison criterion. This allows the comparison criterion to be tailored to the requirements imposed on the mapping.

In this work the average read and write availability value (ARW) criterion is used, which was first presented by the author in [1]. The ARW accumulates the discretized, weighted averages of the $a_r(p)$ and the $a_w(p)$ as shown below:

$$wor \in \{a | a \in \mathbb{R} \wedge 0 \leq a < 1\} \tag{5.9}$$

$$\text{ARW} = wor * \frac{\sum_{i=1}^{100} a_r(i/100)}{100} + (1 - wor) * \frac{\sum_{i=1}^{100} a_w(i/100)}{100} \quad . \tag{5.10}$$

The value *wor* weighs the average $a_r(p)$ against the average $a_w(p)$. This can be used to favor either read or write operations in the selection of the mapping. If not stated differently a *wor* of 0.5 is assumed.

Given the ARW as a comparison criteria, the best mapping can by found by computing the ARW of all mappings and then choosing the mapping with the highest ARW.

To calculate the ARW for a mapping the $a_r(p)$ and the $a_w(p)$ of the mapping need to be computed. As shown in Section 3.3 on page 15 the RQS and the WQS are used to calculate the $a_r(p)$ and the $a_w(p)$. The RQS for the mapping is based on the RQS of the QP that is to be mapped. The same is true for the WQS of the mapping. After a mapping has been applied to RQS or a WQS they are called RQS′ and WQS′. The basic relationship of RQS to the RQS′ is that the mapping turns the RQS used on the LNT into the RQS′ of the used PNT. The RQS′ and the WQS′ are then used to calculate the ARW. This is done for each of the $N!$ mappings. The mapping with the highest ARW is the best mapping.

RQS′ is defined as:

$$\begin{aligned}
\text{RQS}' := \{&(q_1, \{sq_{1,1}, \ldots, sq_{1,m}\}), \ldots, (q_n, \{sq_{n,1}, \ldots, sq_{n,z}\}) \quad | && (5.11)\\
&\wedge q_i = \mathfrak{R}(\mathfrak{M}(q_k, M(G,G')), \text{PNT}), q_k \in \text{RQS} && (5.12)\\
&\wedge sq_{i,j} = \mathfrak{R}(\mathfrak{M}(sq_{o,l}, M(G,G')), \text{PNT}), sq_{o,l} \in \text{RQS} && (5.13)\\
&\wedge (q_i, sq_{i,j}) : sq_{i,j} \supset q_i && (5.14)\\
&\wedge q_i, q_j : q_i \not\supseteq q_j && (5.15)\\
&\wedge (q_i, sq_{i,n}), (q_j, sq_{j,m}) : sq_{i,n} \neq sq_{j,m} && (5.16)\\
&\wedge \nexists (q_i, sq_{i,j}), q_j : sq_{i,j} \supset q_j \wedge |q_i| > |q_j| && (5.17)\\
\}. && (5.18)
\end{aligned}$$

The WQS′ is defined as:

$$\begin{aligned}
\text{WQS}' := \{&(q_1, \{sq_{1,1}, \ldots, sq_{1,m}\}), \ldots, (q_n, \{sq_{n,1}, \ldots, sq_{n,z}\}) \quad | && (5.19)\\
&\wedge q_i = \mathfrak{R}(\mathfrak{M}(q_k, M(G,G')), \text{PNT}), q_k \in \text{WQS} && (5.20)\\
&\wedge sq_{i,j} = \mathfrak{R}(\mathfrak{M}(sq_{o,l}, M(G,G')), \text{PNT}), sq_{o,l} \in \text{WQS} && (5.21)\\
&\wedge (q_i, sq_{i,j}) : sq_{i,j} \supset q_i && (5.22)\\
&\wedge q_i, q_j : q_i \not\supseteq q_j && (5.23)\\
&\wedge (q_i, sq_{i,n}), (q_j, sq_{j,m}) : sq_{i,n} \neq sq_{j,m} && (5.24)\\
&\wedge \nexists (q_i, sq_{i,j}), q_j : sq_{i,j} \supset q_j \wedge |q_i| > |q_j| && (5.25)\\
\}. && (5.26)
\end{aligned}$$

In contrast to the original RQS definition, the definition of the RQS′ consists of mapped elements of the original RQS not the power set of the replicas of the QP. This is shown in Equation 5.12 and Equation 5.13. In both cases, $\mathfrak{M}(q_k, M(G,G'))$ and $\mathfrak{M}(sq_{o,l}, M(G,G'))$, the function $\mathfrak{M}$ maps the replicas of the quorums q_k and $sq_{o,l}$ according to the given mapping $M(G,G')$. Let $q'_k = \mathfrak{M}(q_k, M(G,G'))$ and $sq'_{o,l} = \mathfrak{M}(sq_{o,l}, M(G,G'))$. As explained earlier, the replicas of q'_k must no longer be connected in the given PNT. The same is true for the replicas of $sq'_{o,l}$. Therefore, the function $\mathfrak{R}$ reconnects the replicas in q'_k as well as in $sq'_{o,l}$. The reconnected quorums q_i and $sq_{i,j}$ are then stored in a RQS'. The Equation 5.14 – Equation 5.17 as well as Equation 5.22 – Equation 5.25 have the same purpose as in the original constructor of the RQS and WQS. The make sure that no quorum and there super-sets are sorted correctly, such that the costs and availability of the read and write operations are calculated correctly. Let $(RQS', WQS') = MAP(M(G,G'), RQS, WQS, PNG)$ be the function that calculates the RQS' and WQS' given a mapping $M(G,G')$, an RQS, WQS, and a PNT.

Algorithm 9 on the facing page shows how the best mapping is found. As mentioned earlier, the original QP is no longer of importance. what is needed instead is the RQS and the WQS constructed by the QP. This is reflected in the inputs of the algorithm. Additionally, the algorithm requires a comparison

Algorithm 9: Procedure *bestMapping*

Input: pnt = the PNT to find the best QP for
rqs = the RQS of the QP to map
wqs = the WQS of the QP to map
cmp = the comparison criterion used to compare two QPs
Result: the best mapping according to the criterion

```
1  curBestMapping = ∅
2  perm = (pnt_V)
3  for it ∈ {all permutations of perm} do
4  |  M = ∅
5  |  for i ∈ [0, |it|) do
6  |  |  M = M ∪ (i, it_i)
7  |  end
8  |  (rqs', wqs') = MAP(M, rqs, wqs, pnt)
9  |  if curBestMapping = ∅ ∨ cmp(rqs', wqs') >
   |    cmp(curBestMapping_rqs', curBestMapping_wqs') then
10 |  |  curBestMapping = (rqs', wqs', M)
11 |  end
12 end
13 return curBestMapping
```

criterion, named *cmp*, that is able to order two pairs of a RQS' and a WQS'. If not mentioned otherwise, the previously introduced ARW is used. The algorithm begins, as shown on line 1 in Algorithm 9, by creating a temporary variable to store the currently best mapping. As no mapping has been tested yet, it is the $\emptyset$. As it is unknown which of the $N!$ possible mappings is the under the given comparison criterion, all have to be tested and compared. As the PNT is a GS and it vertices are stored in a set, and sets have no notion of ordering for their elements, an ordering notion has to be created for the permutations. This is done on line 2 in Algorithm 9 where the set of vertices of the PNT is transformed into a tuple. On line 3 in Algorithm 9 the algorithm starts to iterate over all possible permutations of the tuple elements in *perm*. The variable *it* is used as in iteration variable. Next, a mapping called M is constructed. This is easiest explained by an example. Let the GS shown inFigure 5.2 be used as the PNT and $it = (e,b,c,d,a)$. Then the loop shown on line 5 in Algorithm 9 constructs a set $M = \{(0,e),(1,b),(2,c),(3,d),(4,a)\}$ The variable i iterates the values from zero to, but not including, the number of elements in *it*. In each iteration a new tuple gets attached to M On the following two lines, M constructs the RQS$'$ and WQS$'$. The RQS$'$ and the WQS$'$ are constructed adhering to the rules established in Equation 5.11 on the preceding page and Equation 5.19. This RQS$'$ and the WQS$'$ is then compared, using *cmp*, with the currently best mapping. If it is better or if it is the first tested mapping then it gets assigned to the *curBestMapping*

variable, as shown in line 10 in Algorithm 9. This process is repeated for all $N!$ mappings. Finally, the best mapping is returned.

5.3 Example Analysis

In the following, this process, with the example from Section 5.2 on page 47, is illustrated. The RQS of the MCS with five replicas is shown in Table 5.1 on the next page. Starting with the RQ $\{0,1,2\}$, this RQ gets mapped by function $\mathfrak{M}$ to the replicas $\{c,d,e\}$ and the mapping $\{(0,e),(1,d),(2,c),(3,b),(4,a)\}$. The MCS was purposefully chosen for this example as ARW for $N1$ mappings are equal. Therefore, only one mapping has to be considered for the MCS and it does not matter which. This allows, to show how the process works and also have the best possible mapping as the result of the process. In the given PNT, as shown in Figure 5.2, the replica with ID e is not connected to the other two replicas. Therefore, function $\mathfrak{R}$ has to add the replicas with ID 3 in the LNT and ID b in the PNT to the set of mapped replicas, to reconnect the replicas. The resulting RQ $\{b,c,d,e\}$ is inserted in the RQS$'$. Repeating the same steps for the RQ $\{0,1,3\}$ the same RQ $\{b,c,d,e\}$ is obtained. As all sets of replicas in the RQS$'$ have to be unique, this second instance of this set of replicas is discarded. After repeating this for all the q_i and $sq_{o,l}$ elements of the original RQS, the RQS$'$ as shown in Table 5.2 on the facing page is obtained. The original RQS consists of 10 q_i elements and 6 $sq_{o,l}$ elements. The RQS$'$ construct by the mapping consists of 6 q_i elements and 6 $sq_{o,l}$ elements. This reduction is due to the fact that reconnecting replicas in the PNT adds replicas to the quorum which in turn leads duplications. But as the equations for the RQS, WQS, RQS$'$, and WQS$'$ show duplicates are not allowed. This leads to a reduction of the $a_r(p)$ of the mapped MCS as shown in Figure 5.4 on page 54, even though the mapping shown has the highest ARW of all possible mappings. For $N = 5$ the read and write operation of the MCS uses the same quorums. Therefore, only the read operation is plotted in Figure 5.4, Figure 5.5. Figure 5.5 on page 54 shows a close-up of Figure 5.4 where $p \geq 0.8$. As mentioned earlier, this view of the analysis is important as empirical evidence suggests that usually replicas have an availability of more than 80%. Especially, the close-up shows the difference in the $a_r(p)$ the mapping makes. For $p = 0.8$ there is nearly a 10% difference. It is important to note here that the mapping does not worsen the QP, but strengths the assumptions about the overall system. Therefore, the mapping $a_r(p)$, the $a_w(p)$ are lower, but the results are more accurate.

$$
\begin{aligned}
\mathrm{RQS} = \{ & \\
& (\{0,1,2\},\{\{0,1,2,3\},\{0,1,2,4\},\{0,1,2,3,4\}\}), \\
& (\{0,1,3\},\{\{0,1,3,4\}\}), \\
& (\{0,1,4\},\{\}), \\
& (\{0,2,3\},\{\{0,2,3,4\}\}), \\
& (\{0,2,4\},\{\}), \\
& (\{0,3,4\},\{\}), \\
& (\{1,2,3\},\{\{1,2,3,4\}\}), \\
& (\{1,2,4\},\{\}), \\
& (\{1,3,4\},\{\}), \\
& (\{2,3,4\},\{\}) \\
& \quad \}
\end{aligned}
$$

Table 5.1: The RQS of the MCS that is to be mapped to the PNT in Figure 5.2.

$$
\begin{aligned}
\mathrm{RQS}' = \{ & \\
& (\{a,b,c\},\{\{a,b,c,d\},\{a,b,c,e\},\{a,b,c,d,e\}\}), \\
& (\{a,b,d\},\{\{a,b,d,e\},\}), \\
& (\{a,b,e\},\{\}), \\
& (\{a,c,d\},\{\{a,c,d,e\},\}), \\
& (\{b,c,d\},\{\{b,c,d,e\},\}), \\
& (\{b,d,e\},\{\}) \\
& \quad \}
\end{aligned}
$$

Table 5.2: The RQS′ of the MCS after it is mapped to the PNT in Figure 5.2.

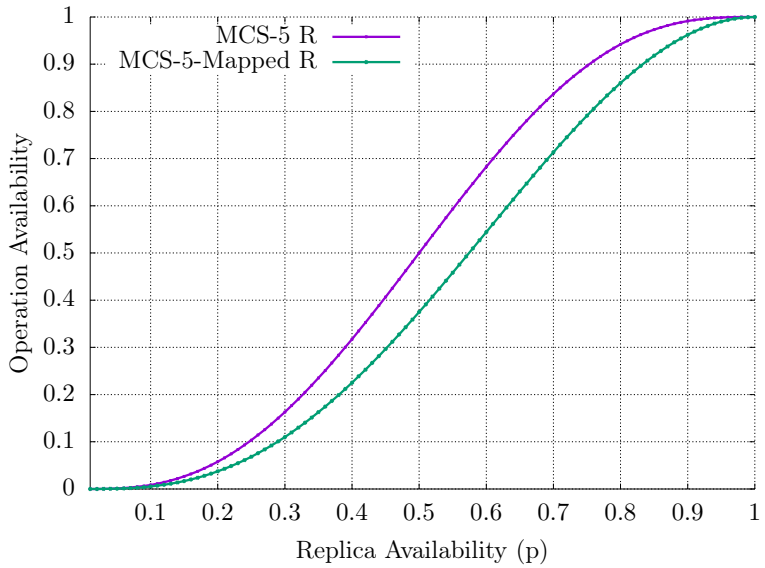


Figure 5.4: The read (R) operation availability of the mapped and unmapped MCS.

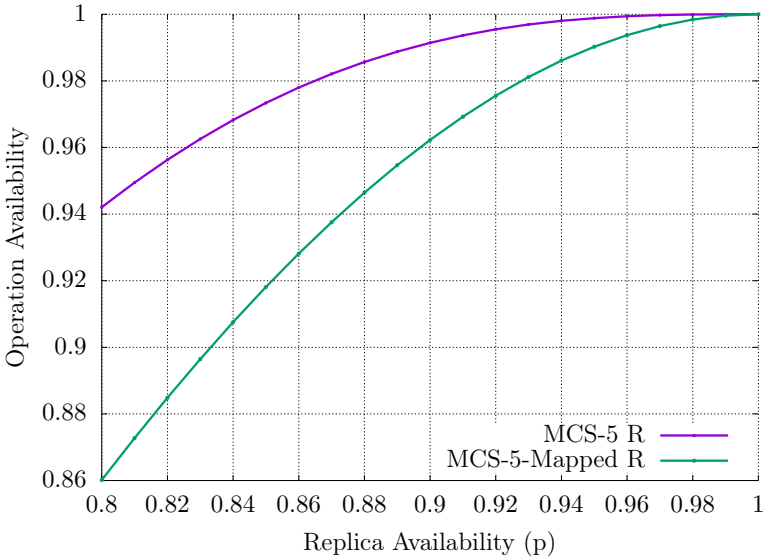


Figure 5.5: The read (R) operation availability of the mapped and unmapped MCSwith $p \geq 0.8$.

5.4 Evalutation

To evaluate the impact of the mapping on the availability and the costs measures in a more general case, then the in the previous section, the GSs used as a PNTs have to be analyzed. For instance, the GS in Figure 5.7 on the next page will yield mappings whose availability and cost measurements equal to those one's of the QP mapped to it, because all of the vertices are connected to each other. The Figure 5.6 on the other hand will have significantly lower availability and higher cost measurements, than the original QP, because the removal of one vertex could partition the GS.

Therefore, the idea behind this analysis is to generate many non-isomorphic graphs with the same number of vertices and then use these as PNTs. The $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ values are then used within statistical methods to analyze the impact of the mapping. The idea is that many randomly generated PNTs will give a good indication of what the influence of the mapping process on the cost and availability of the read and write operation is.

Assuming, the QPs require nine replicas, an ideal analysis would test the mappings to all possible GSs with nine vertices. This is impossible as the number of possible GSs that are non-isomorphic is infinite even with a single vertex, the GSs used for the evaluation have to be restricted. Figure 5.8 on the next page shows how to construct an infinite amount of GSs with one replica. Each new GS is constructed by adding a new self loop. In Figure 5.8 adding an infinite amount of self loops is represented by the three dots.

The first restriction is to only allow simple connected GSs. Figure 5.9 on the following page shows a non-simple, non-connected GS. The GS in Figure 5.9 is a non-simple GS, because it has a self loop on vertex a and multiple edges between vertices b and c. Additionally, is a non-connected GS, as there is no edge between vertex d and any other vertex of the GS. Figure 5.10 on page 57 shows the GS from Figure 5.9 transformed into a simple, connected GS. For simple, connected GSs the number of possible GSs can be computed given the number of vertices. Table 5.3 on page 57 shows the number of non-isomorphic, simple, connected GS based on the number of vertices in a GS. These values were computed for this work as no reference could be found in the literature. From the rapidly growing number of GSs it can be concluded that evaluating the complete state space becomes infeasible quickly. For instance, evaluating all mappings for the 29337 GSs with seven nodes would require to test $147858480 = 29337 \cdot 7!$ mappings.

Therefore, sampling is used to draw conclusions on the complete population. For GSs with eight and nine vertices 255 simple, randomly connected, non-isomorphic graphs are created. For all the GSs with eight vertices, the mapping with the

Figure 5.6: A GS not well used to serve as a PNT.

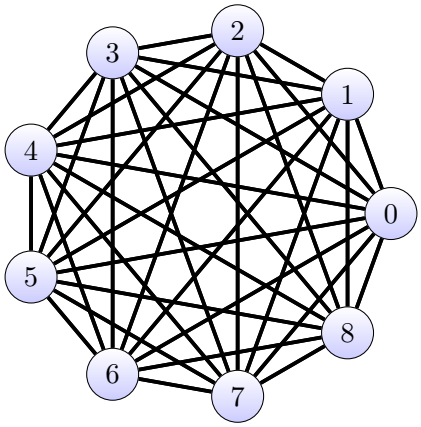


Figure 5.7: A GS well used to serve as a PNT.

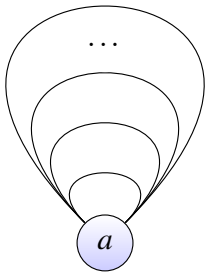


Figure 5.8: An example of creating an infinite amount of GSs with on vertex.

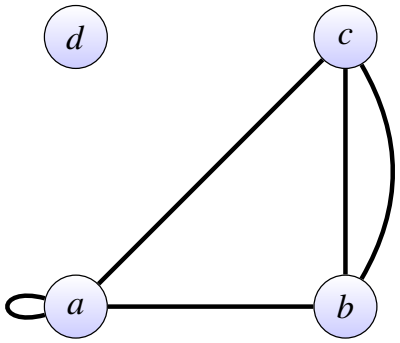


Figure 5.9: A non-simple, non-connected GS

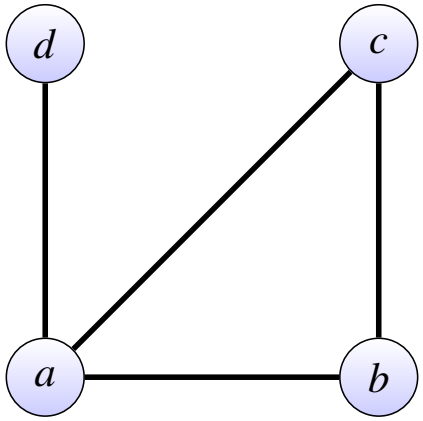


Figure 5.10: A simple connected GS

Number of vertices	Number of GSs
1	1
2	1
3	2
4	6
5	35
6	603
7	29337

Table 5.3: The number of non-isomorphic, simple, connected GS based on the number of vertices in the GS.

highest ARW for the MCS with eight replicas, the GP with a 2×4 LNT and a 4×2 LNT, as well as for the TLP with a 2×4 LNT and a 4×2 LNT is found. For all the GSs with nine vertices, the mapping with the highest ARW for the MCS with nine replicas, the GP with a 3×3 LNT, as well as for the TLP with a 3×3 LNT is found. PNTs with eight and nine replicas were chosen as mappings with nine replicas is the practical limit that could be analyzed in reasonable time [1]. The next square LNT would be 4×4 for which a mapping is $\approx 60 * 10^6$ more complex to analysis. Table 5.4 shows an exemplary result of the $a_r(p)$ of the mappings with the highest ARW of a QP mapped to 255 different graphs. Each column labeled 0 to 254 represents one mapping. Each row shows the operation availabilities of each mapping given the replicas have a p-value as shown in the first column. Each cell shows the availability of the operation. From such a tables the minimum, the average, the median, the 0.25 quantile, the 0.75 quantile and the maximum operation availability are computed. Each following figure either shows the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, or the $c_w(p)$ property of a specific QP. In each figure 101 boxplots are shown. The boxplots are aligned to the p-values. The p-values start at 0.0 and are incremented to 1.0 in 0.01 steps. The p-value of 1.0 is included in the figure. A boxplot as shown in Figure 5.11 on the facing page represents six values. The bottom whisker of the boxplot represents the minimum value of the measured property. The bottom line of the box represents the 0.25 quantil value of the measured property. The black line represents the median of the measured property. The red line represents the average of the measured property. The top line of the box represents the 0.75 quantil value of the measured property. The top whisker of the boxplot represents the maximum value of the measured property. These values are aggregated from the mappings of the tested QPs to the 255 GSs as shown in Table 5.4. The light blue line in each figure shows the original value before the mapping as a reference for comparison. Additionally, to the box plots

[1] Reasonable in this case is days.

p \ Ids	0	1	2	…	254
0.00	0.0000000	0.0000000	0.0000000	…	0.0000000
0.01	0.0004969	0.0004989	0.0004963	…	0.0005069
0.02	0.0019750	0.0019750	0.0019750	…	0.0019750
0.03	0.0044203	0.0044140	0.0044723	…	0.0044001
…	…	…	…	…	…
0.97	0.9999099	0.9999179	0.9999183	…	0.9999181
0.98	0.9999759	0.9999758	0.9999698	…	0.9999778
0.99	0.9999969	0.9999968	0.9999968	…	0.9999979
1.00	1.0000000	1.0000000	1.0000000	…	1.0000000

Table 5.4: Table showing the $a_r(p)$ of a QP mapped to 255 graphs.

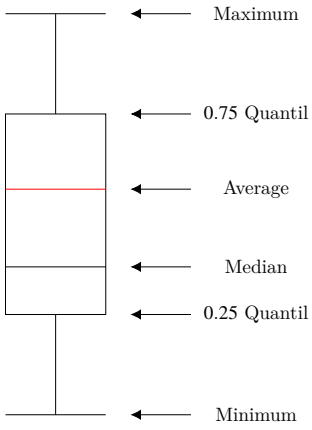


Figure 5.11: A boxplot

the standard deviation (SD) and the median absolute deviation (MAD) for the availabilities are shown. Where SD is defined as:

$$SD = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{5.27}$$

, $\bar{x}$ is the average, and N is the number of mappings. The MAD is defined as:

$$MAD = \frac{1}{N}\sum_{i=0}^{N}|x_i - \tilde{x}| \tag{5.28}$$

where $\tilde{x}$ is the median, and $x_1, \ldots, x_N$ are elements in a tuple were the elements are sorted. The MAD was also evaluated as its value is on skewed by outliers as easily as the SD, but when plotted the MAD sometimes appears jerky.

Mapping evaluation of the MCS with eight replicas: Figure 5.13 on page 64 to Figure 5.18 on page 69 show the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ of the MCS, with eight replicas, mapped to 255 different GSs. In Figure 5.13 and Figure 5.14 on page 65 it can be seen that the operation availability of the bulk of the mappings is about five to ten percent below that of the unmapped MCS QP. For small $p < 0.4$ and big $p > 0.8$ values the difference shrinks. This is also reflected in Figure 5.15 on page 66 and Figure 5.16 on page 67. There, the SD and the MAD of the mapped read and write availability is shown. These figures show how read and write availabilities of the 255 mapping deviate. They confirm the higher spread around $0.4 < p < 0.8$. Figure 5.17 on page 68 and Figure 5.18 show the $c_r(p)$ and the $c_w(p)$ of the mapped MCS. The $c_r(p)$ of the mapped MCS

is always four, the same cost as for the unmapped MCS [2]. The $c_w(p)$ is also the same between the mappings and the original MCS.

Mapping evaluation of the GP with 2×4 replicas: Figure 5.19 on page 70 to Figure 5.24 on page 75 show the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ of the GP, with 2×4 replicas, mapped to 255 different GSs. The difference between the mapped GS and the unmapped GS are less severe in comparison to the MCS. The bulk of the mappings are close to the original read and write availability of the GP, which can be seen in Figure 5.19 and Figure 5.20 on page 71. This is also shown in the SD and the MAD of the $a_r(p)$ and the $a_w(p)$ as shown in Figure 5.21 on page 72 and Figure 5.22 on page 73. Their values are much lower than those of the MCS with eight replicas. The $c_r(p)$ and the $c_w(p)$ are shown in Figure 5.23 on page 74 and Figure 5.24. The $c_r(p)$ is always 2, also for the mapping. The $c_w(p)$ is more interesting. The original QP has a $c_w(p)$ of 5, as can be derived from its operation. The worst mapping has a $c_w(p)$ of six[3].

Mapping evaluation of the GP with 4×2 replicas: Figure 5.25 on page 76 to Figure 5.30 on page 81 show the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ of the GP, with 2×4 replicas, mapped to 255 different GSs. The unmapped QP with its replicas arranged in a 4×2 grid are even more closely mirrored by its mappings. The $a_r(p)$ shown in Figure 5.25 is similar to the $a_r(p)$ of the QP with a 2×4 grid. The same is true for the $a_w(p)$ shown in Figure 5.26 on page 77. The SD and the MAD of the $a_r(p)$ and the $a_w(p)$ of the GP with a 4×2 grid is shown in Figure 5.27 on page 78 and Figure 5.28 on page 79. In comparison to the GP with a 2×4 grid, the SD and MAD is lower. Interestingly, the MAD is 0 for all p. This is due to the density of the results as well as the construction of the MAD itself. Figure 5.23 and Figure 5.24 show the $c_r(p)$ and the $c_w(p)$. The $c_r(p)$ is basically four with some small outliers. The $c_w(p)$ is five for all mappings and for all p.

Mapping evaluation of the TLP with 2×4 replicas: Figure 5.31 on page 82 to Figure 5.36 on page 87 show the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ of the TLP, with 2×4 replicas, mapped to 255 different GSs. In the Figure 5.31 the $a_r(p)$ of the mappings is very close to the $a_r(p)$ of the original TLP. The spread of the mapped $a_w(p)$ is bigger than that of the original TLP, as can be seen in Figure 5.32 on page 83. The bulk of the mappings are still close to the original TLP $a_w(p)$. The SD and the MAD of the $a_r(p)$ and the $a_w(p)$ of the TLP is shown in Figure 5.33 on page 84 and Figure 5.34 on page 85. Again, the MAD is zero and the SD is rather low, as most mappings have the same $a_r(p)$ and the same $a_w(p)$. The $c_r(p)$ is shown in Figure 5.35 on page 86. The difference between the unmapped

[2] The light blue line overlaps with the lines of the boxplot. The blue line is drawn last and, therefore, overlays the different colored lines.

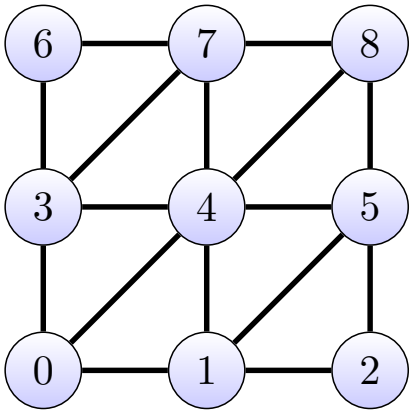
[3] For $p = 0.0$ in Figure 5.24, the cost is zero, the value shown is an error due to floating point math inaccuracies.

TLP and the mapped TLP is small. Especially the difference between the mapped and unmapped read operation is very small, this is because is comparatively easy to find a vertical or horizontal path of two elements. For some mappings the $c_r(p)$ is lower than that of the original TLP. This is due to the restrictions of the triangular lattice grid used by the TLP as shown in Figure 5.12a on the following page. If replica with ID 4 is not available in Figure 5.12a the cheapest quorum $\{0,4,8\}$ is no longer available, leading to increased costs. In Figure 5.12b on the next page there is another path between replicas 0 and 8 that requires only one more replica. This path is $(0,3,8)$. The same phenomena appears for the $c_w(p)$ as shown in Figure 5.36. But overall, the cost per operation of the mapped TLP for both the $c_r(p)$ and the $c_w(p)$ does not change much in comparison to the original TLP.

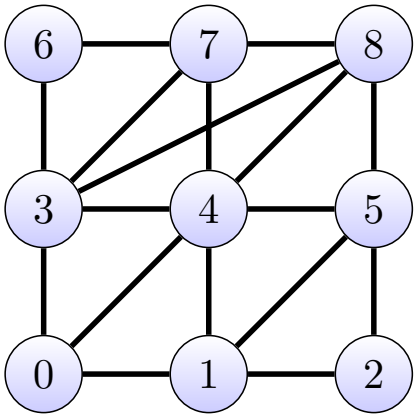
Mapping evaluation of the TLP with 4×2 replicas: Figure 5.37 on page 88 to Figure 5.42 on page 93 show the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ of the TLP, with 4×2 replicas, mapped to 255 different GSs. The difference between the $a_r(p)$ of the mapped TLP and the unmapped TLP is again very small. This observation is strengthened by the very small SD and MAD shown in Figure 5.39 on page 90. The $a_w(p)$ as shown in Figure 5.38 on page 89 has some more outliers in comparison to the $a_r(p)$, but overall shows a similar picture, as confirmed by the SD and MAD shown in Figure 5.40 on page 91. The $c_r(p)$ as shown in Figure 5.35 and the $c_w(p)$ as shown in Figure 5.35 behave pretty much the same as for the TLP with the 2×4 triangular lattice. The difference can be reduced to the change in LNT geometry.

Mapping evaluation of the MCS with nine replicas: Figure 5.43 on page 94 to Figure 5.46 on page 97 show the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ of the MCS, with nine replicas, mapped to 255 different GSs. The $a_r(p)$ of the mapped MCS, as shown in Figure 5.43, with nine replicas behaves similar to the $a_r(p)$ of the MCS with eight replicas. But the majority of the $a_r(p)$ of the mapping is closer to the unmapped $a_r(p)$. The mapped $a_w(p)$, displayed in Figure 5.43, shows the same availabilities as the $a_r(p)$, due to the symmetric nature of the MCS. This symmetry of the read and write operation is also reflected in the SD and MAD plots, as shown in Figure 5.44 on page 95 and Figure 5.45 on page 96, making them equal for the two operations. As the MCS with nine replicas uses the same RQs and WQs, the cost measurements shown in Figure 5.46 and Figure 5.46 are equally set to five.

Mapping evaluation of the GP with nine replicas: Figure 5.47 on page 98 to Figure 5.52 on page 103 show the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ of the GP, with 3×3 replicas, mapped to 255 different GSs. The difference between the $a_r(p)$ of the bulk of the mapping and the original QPs $a_r(p)$ is relatively small, especially for $0.2 < p < 0.6$ as shown in Figure 5.47. Figure 5.48 on page 99 shows



(a) GS used by the TLP with 9 replicas.



(b) GS used by a mapped TLP with 9 replicas.

Figure 5.12: GSs where the mapped QP has smaller costs per operation.

the $a_w(p)$ of the mapped QP in comparison to the unmapped QP. As with the $a_r(p)$ the differences in the $a_w(p)$ are small, especially for $0.2 < p < 0.6$. The SD and the MAD of both the $a_r(p)$ and the $a_w(p)$ are relative big in comparison to the other tests. The $c_r(p)$, displayed in Figure 5.51 on page 102, shows little difference between the mappings and the original GP. The same holds for the $c_w(p)$ as shown in Figure 5.52.

Mapping evaluation of the TLP with nine replicas: Figure 5.53 on page 104 to Figure 5.58 on page 109 show the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ of the TLP, with 3×3 replicas, mapped to 255 different GSs. The difference between the $a_r(p)$ of the mapped TLP and the unmapped TLP is rather small with some exceptions. The $a_r(p)$ is shown in Figure 5.53. The difference between the $a_w(p)$ of the mapped TLP and the unmapped TLP is also small, similar to the $a_r(p)$ result. The $a_w(p)$ is shown in Figure 5.54 on page 105. The spread between the mapped $a_r(p)$ and mapped $a_w(p)$ values is small as shown in Figure 5.55 on page 106 and Figure 5.56 on page 107. Most of the mapped $c_r(p)$ and $c_w(p)$ values are close to the value unmapped TLP, as shown in Figure 5.57 on page 108 and Figure 5.58.

Mapping evaluation conclusion: Looking at the evaluation, it can be said that there is a difference between the operation availabilities, the operation costs of the QPs in their unmapped state in comparison to their mapped state. In the general case, the $a_r(p)$ and the $a_w(p)$ of the QPs decreases when mapped. This difference highly depends on the LNT and the PNT, as well as the QP that is mapped. LNTs with less edges are more easily mapped to a PNT as shown for the GP. The MCS, with its complete graph as LNT, is not as easily mapped.

As said in the introduction, by evaluating the PNT and not the LNT as a communications network, it is shown in this analysis the operation availability

decreases and the operation cost raise. Using the presented mapping approach on the other hand decreases the strength of the assumptions made by QPs.

No analyses with ten or more replicas were carried out, due to the computational complexity of these analyses. PNTs with ten vertices are already ten times more complex to analysis, PNTs with eleven vertices are 110 times as complex.

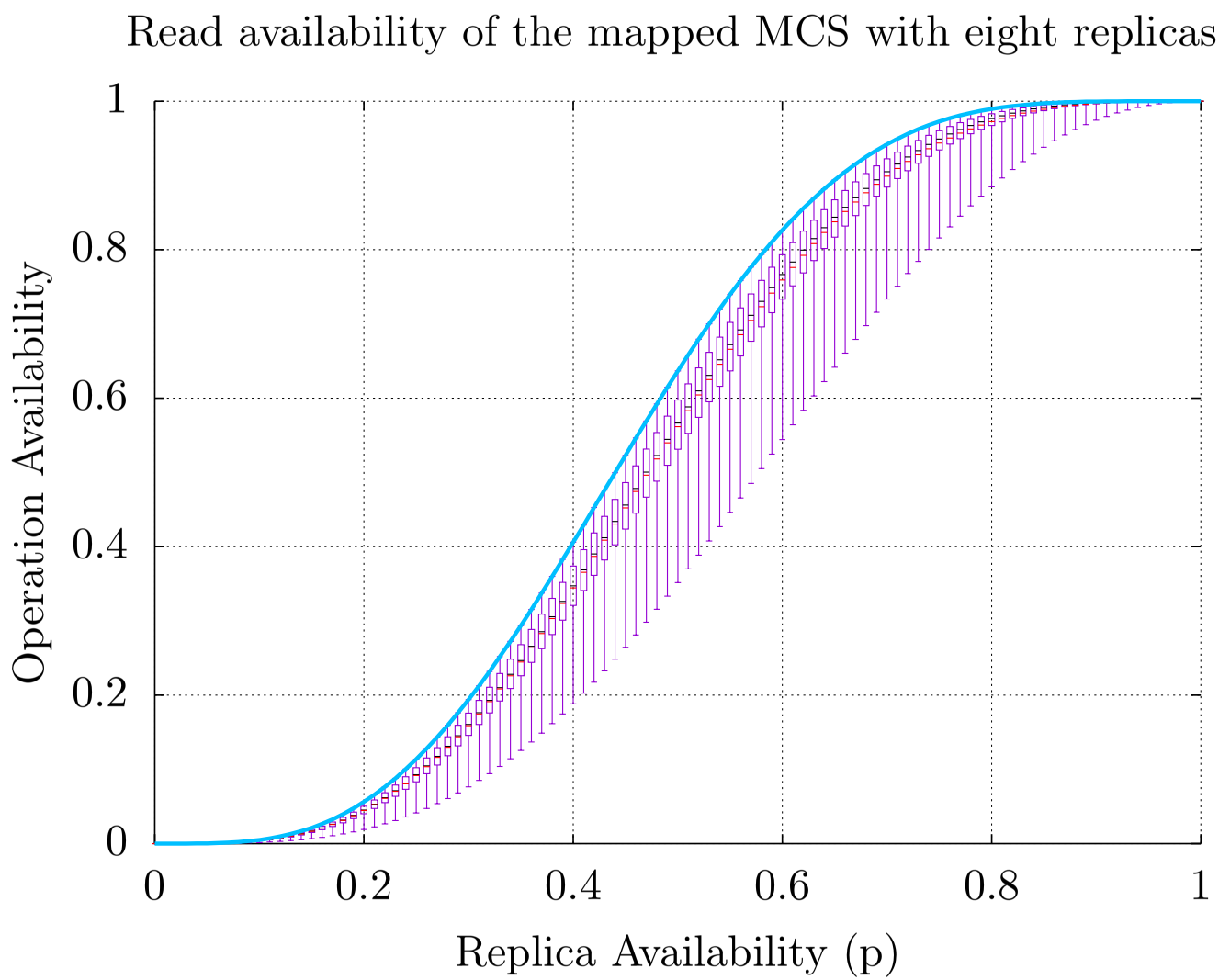


Figure 5.13: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ of the MCS with eight replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_r(p)$ of the MCS with eight replicas in its unmapped state.

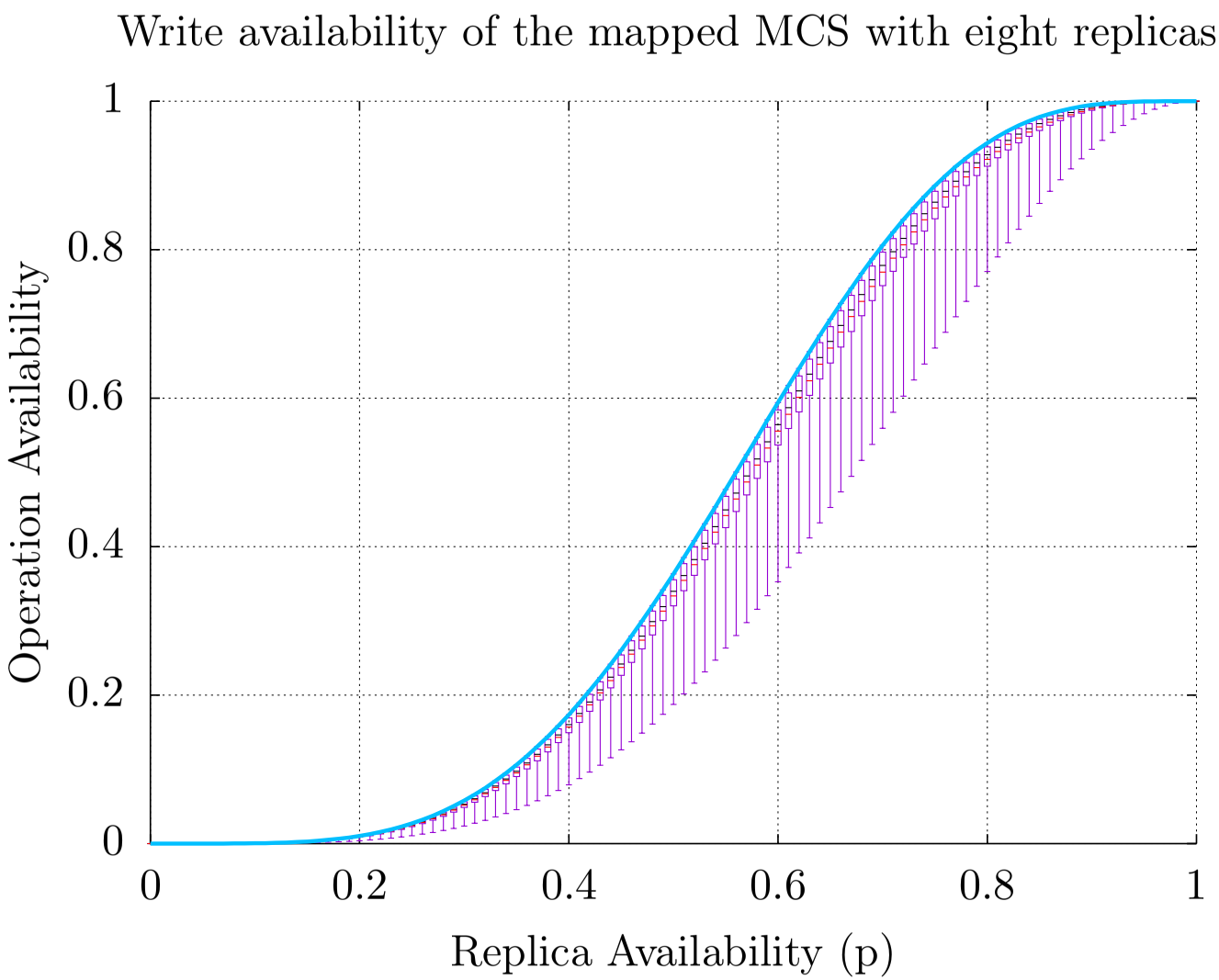


Figure 5.14: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_w(p)$ of the MCS with eight replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_w(p)$ of the MCS with eight replicas in its unmapped state.

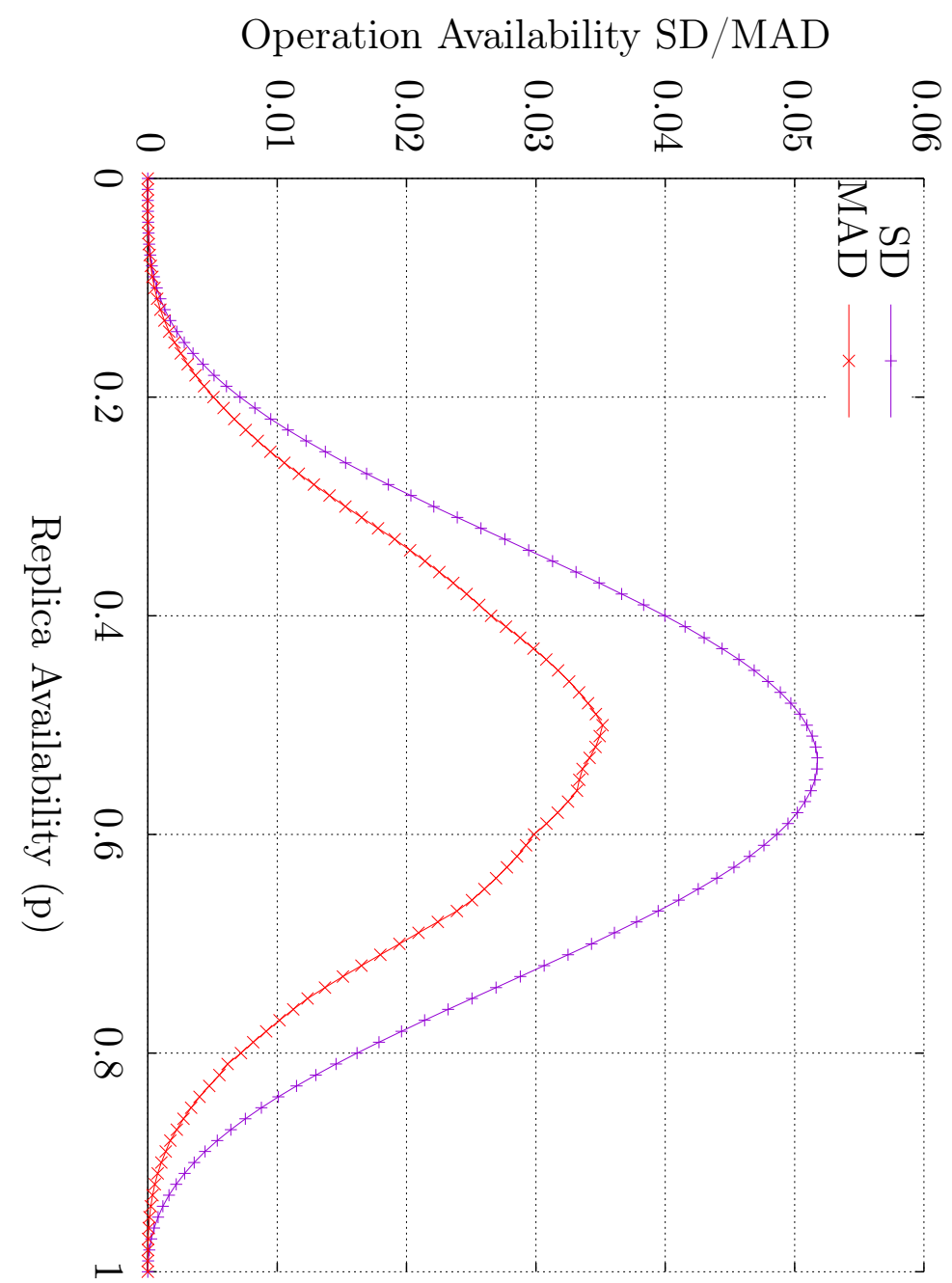


Figure 5.15: The minimal SD and MAD of the $a_r(p)$ of the MCS with eight replicas mapped to 255 simple, non-isomorphic GSs.

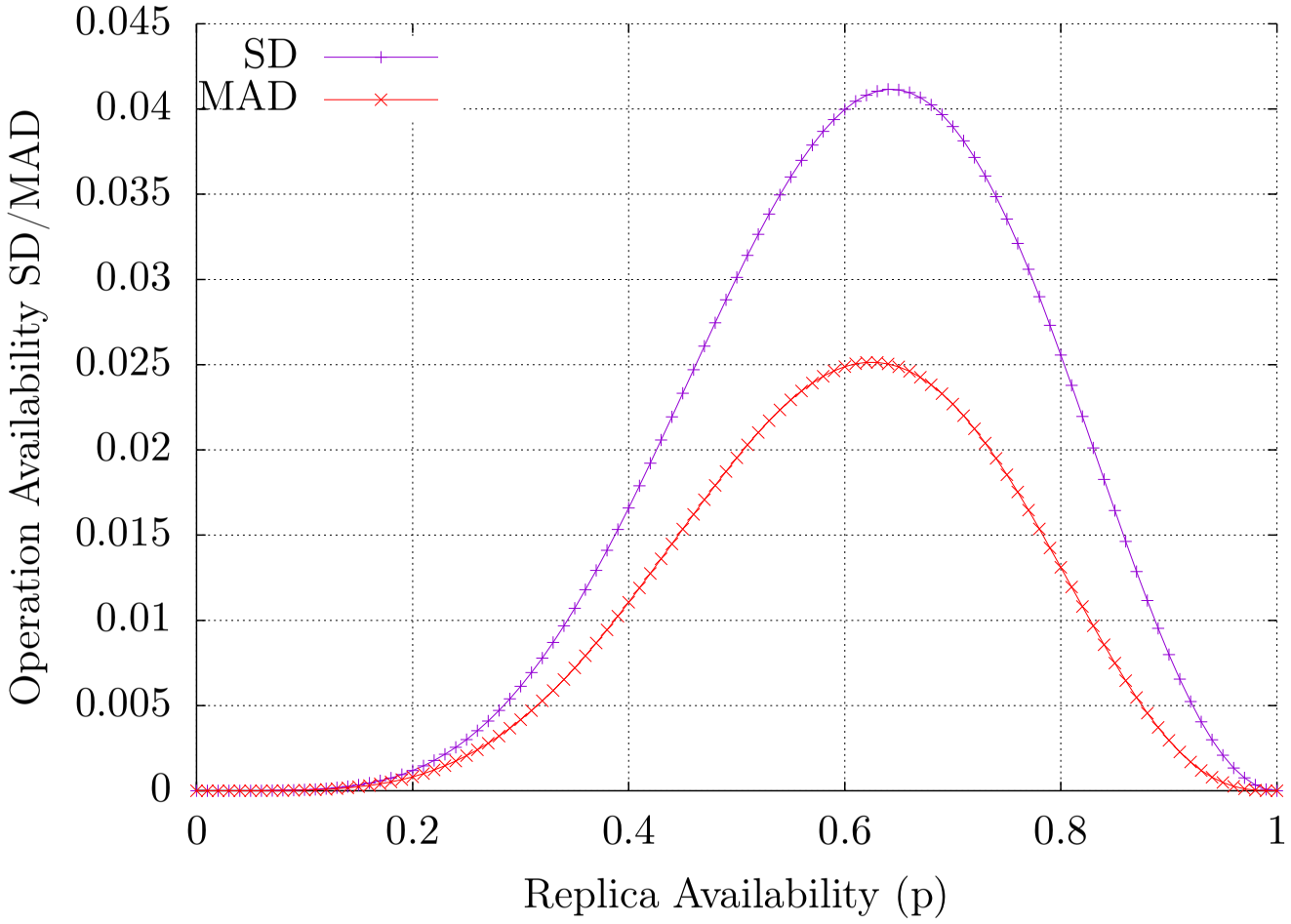


Figure 5.16: The minimal SD and MAD of the $a_w(p)$ of the MCS with eight replicas mapped to 255 simple, non-isomorphic GSs.

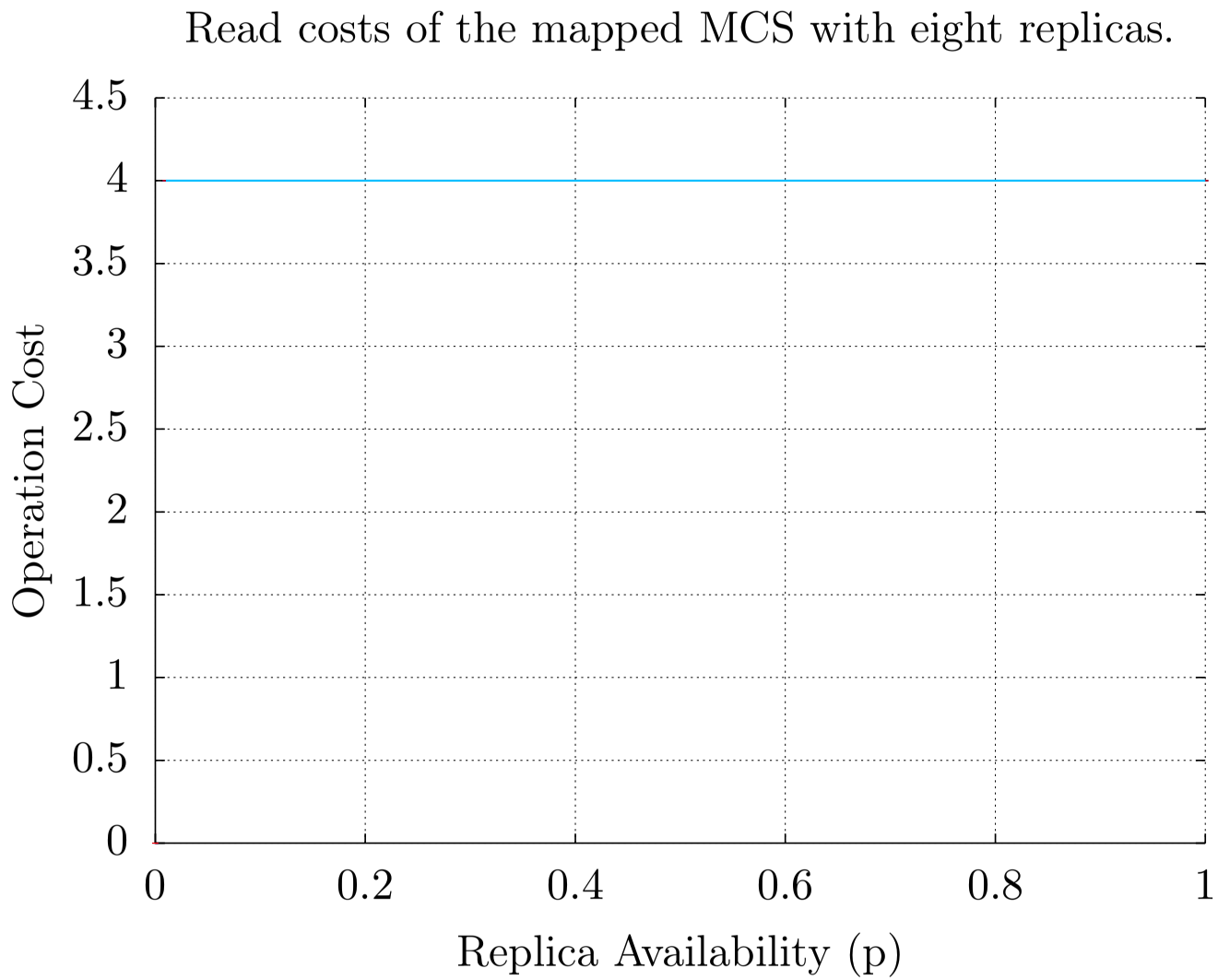


Figure 5.17: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ of the MCS with eight replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_r(p)$ of the MCS with eight replicas in its unmapped state. All noted values are four, and therefore overlap in the plot.

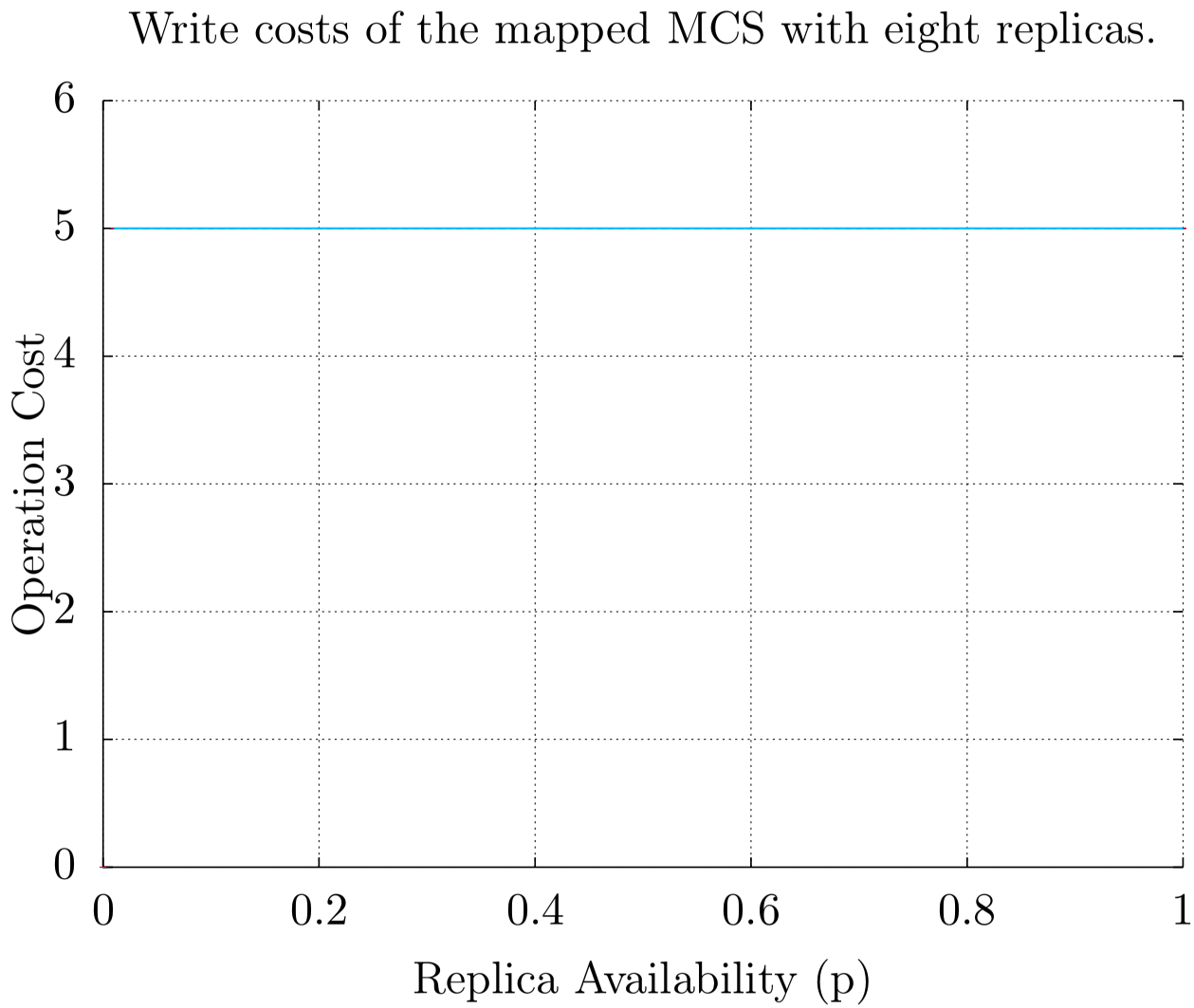


Figure 5.18: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_w(p)$ of the MCS with eight replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_w(p)$ of the MCS with eight replicas in its unmapped state. All noted values are five, and therefore overlap in the plot.

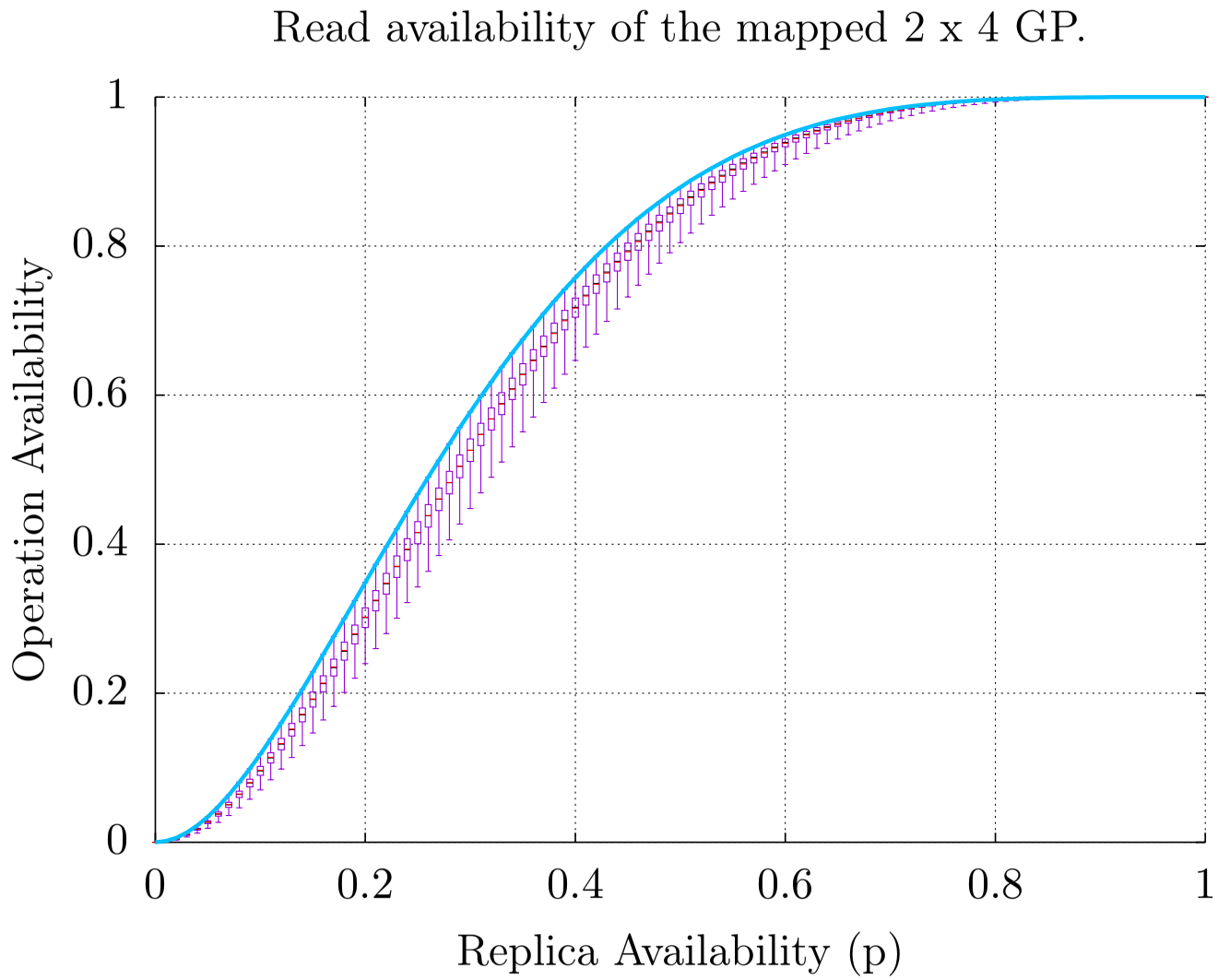


Figure 5.19: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ of the GP with 2×4 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_r(p)$ of the GP with 2×4 replicas in its unmapped state.

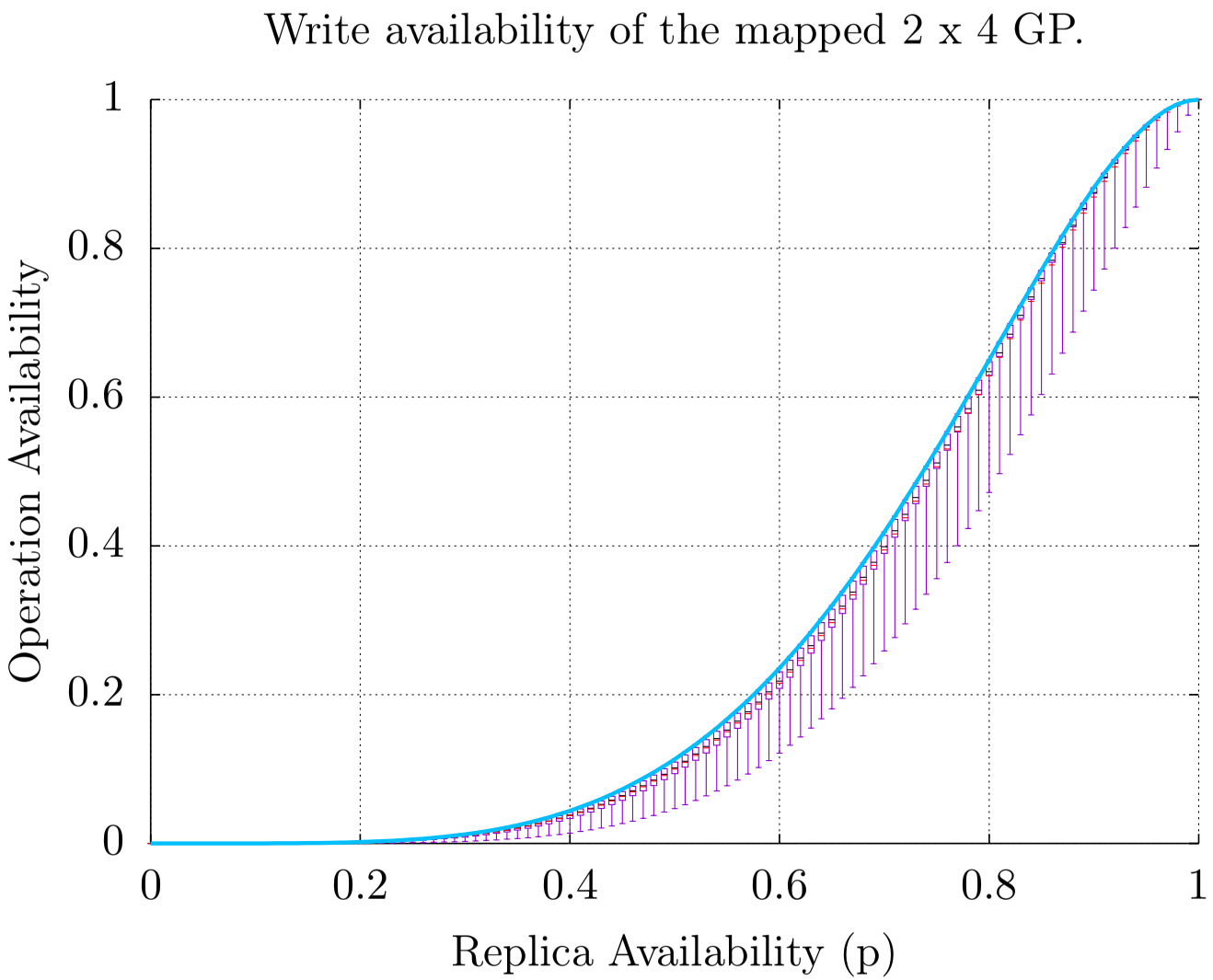


Figure 5.20: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_w(p)$ of the GP with 2×4 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_w(p)$ of the GP with 2×4 replicas in its unmapped state.

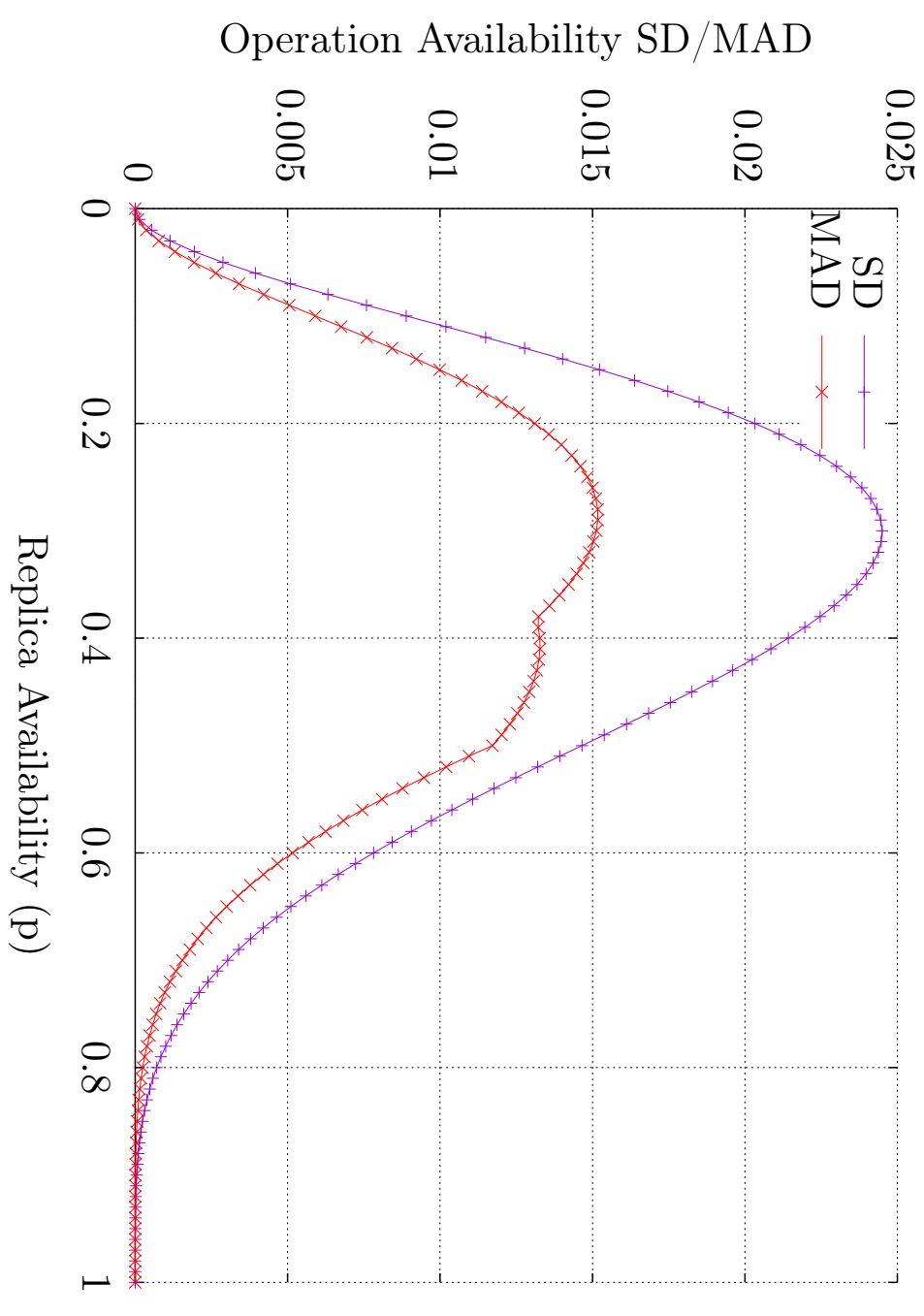


Figure 5.21: The minimal SD and MAD of the $a_r(p)$ of the GP with 2×4 replicas mapped to 255 simple, non-isomorphic GSs.

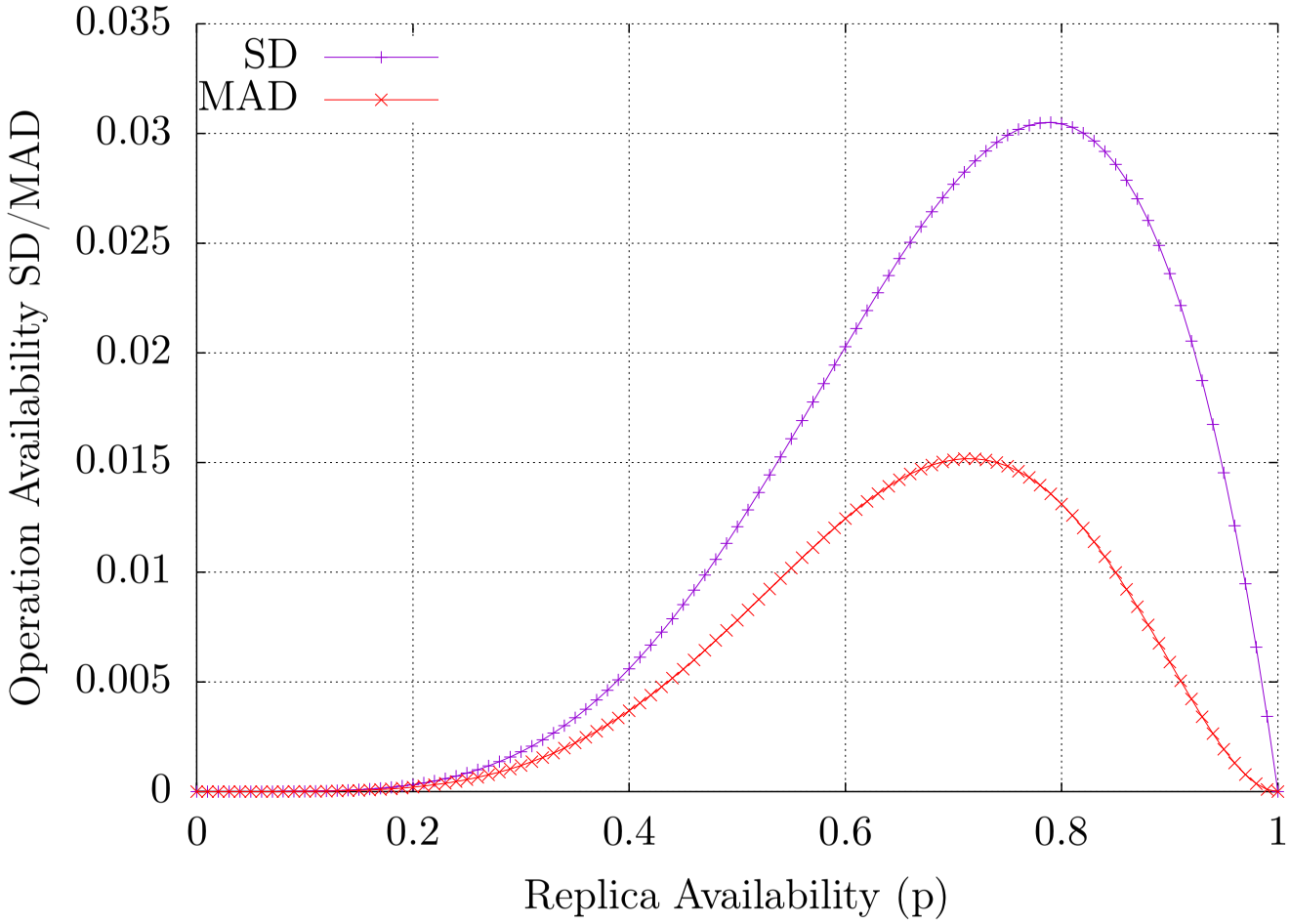


Figure 5.22: The minimal SD and MAD of the $a_w(p)$ of the GP with 2×4 replicas mapped to 255 simple, non-isomorphic GSs.

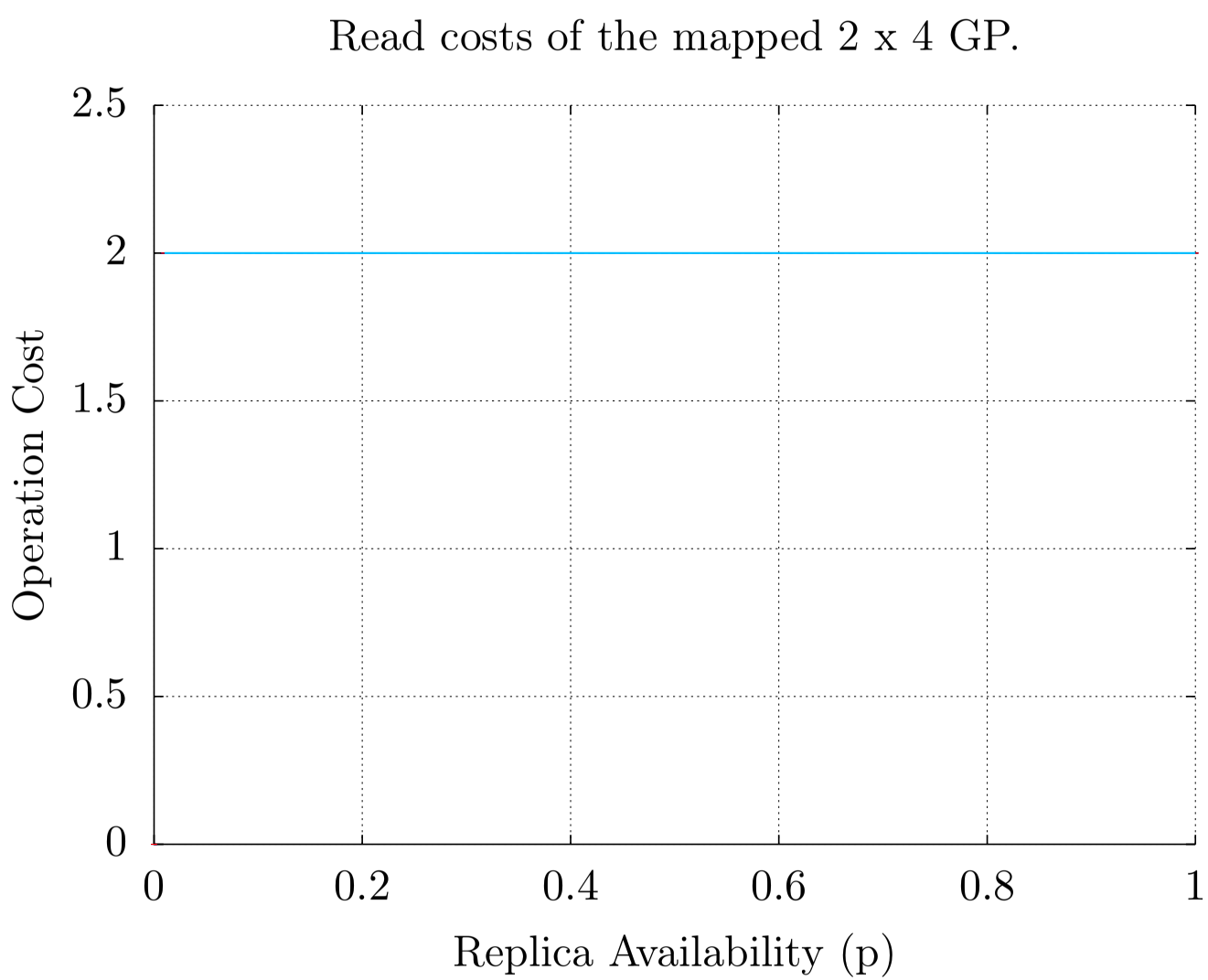


Figure 5.23: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ of the GP with 2×4 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_r(p)$ of the GP with 2×4 replicas in its unmapped state. All noted values are two, and therefore overlap in the plot.

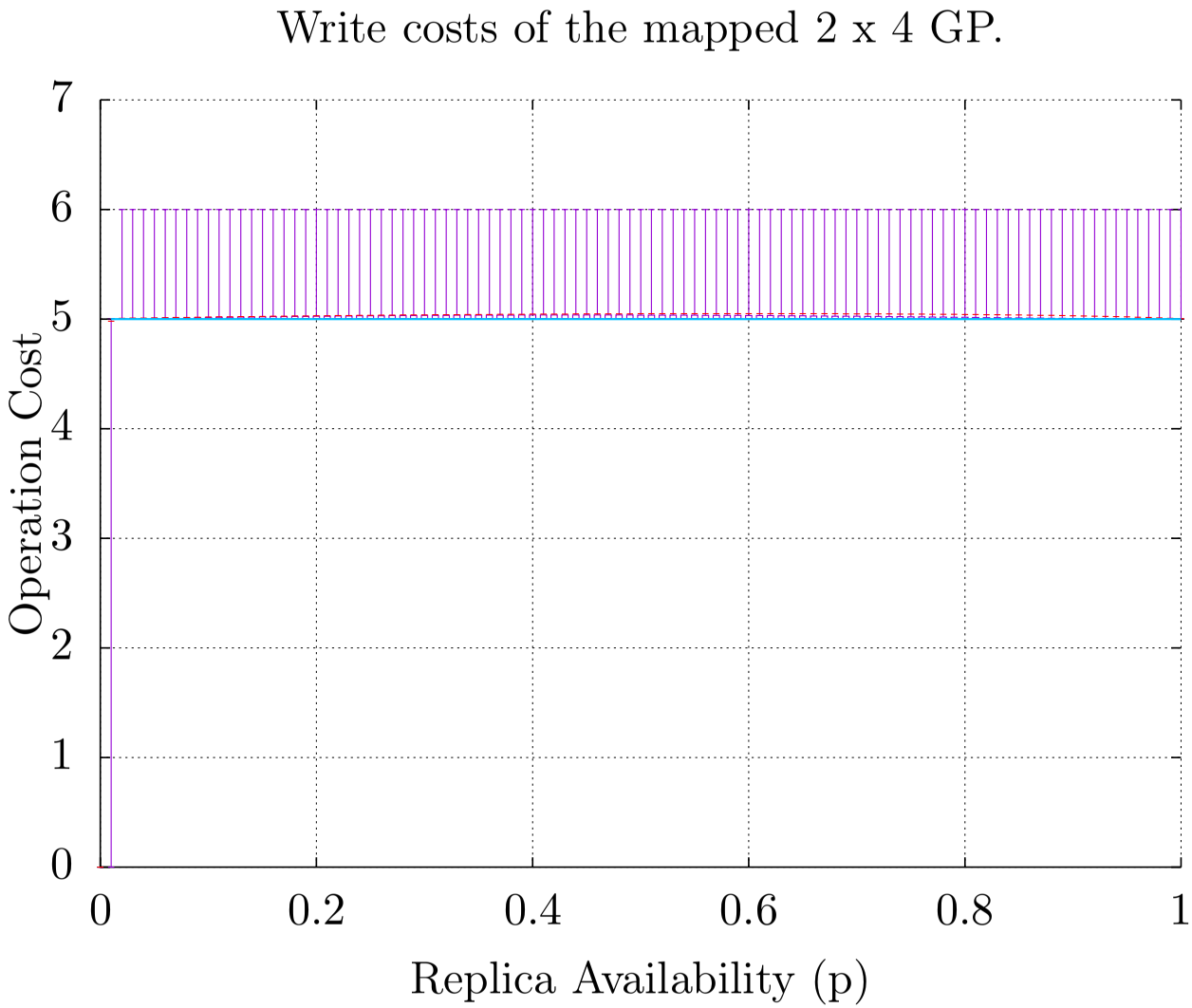


Figure 5.24: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_w(p)$ of the GP with 2×4 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_w(p)$ of the GP with 2×4 replicas in its unmapped state.

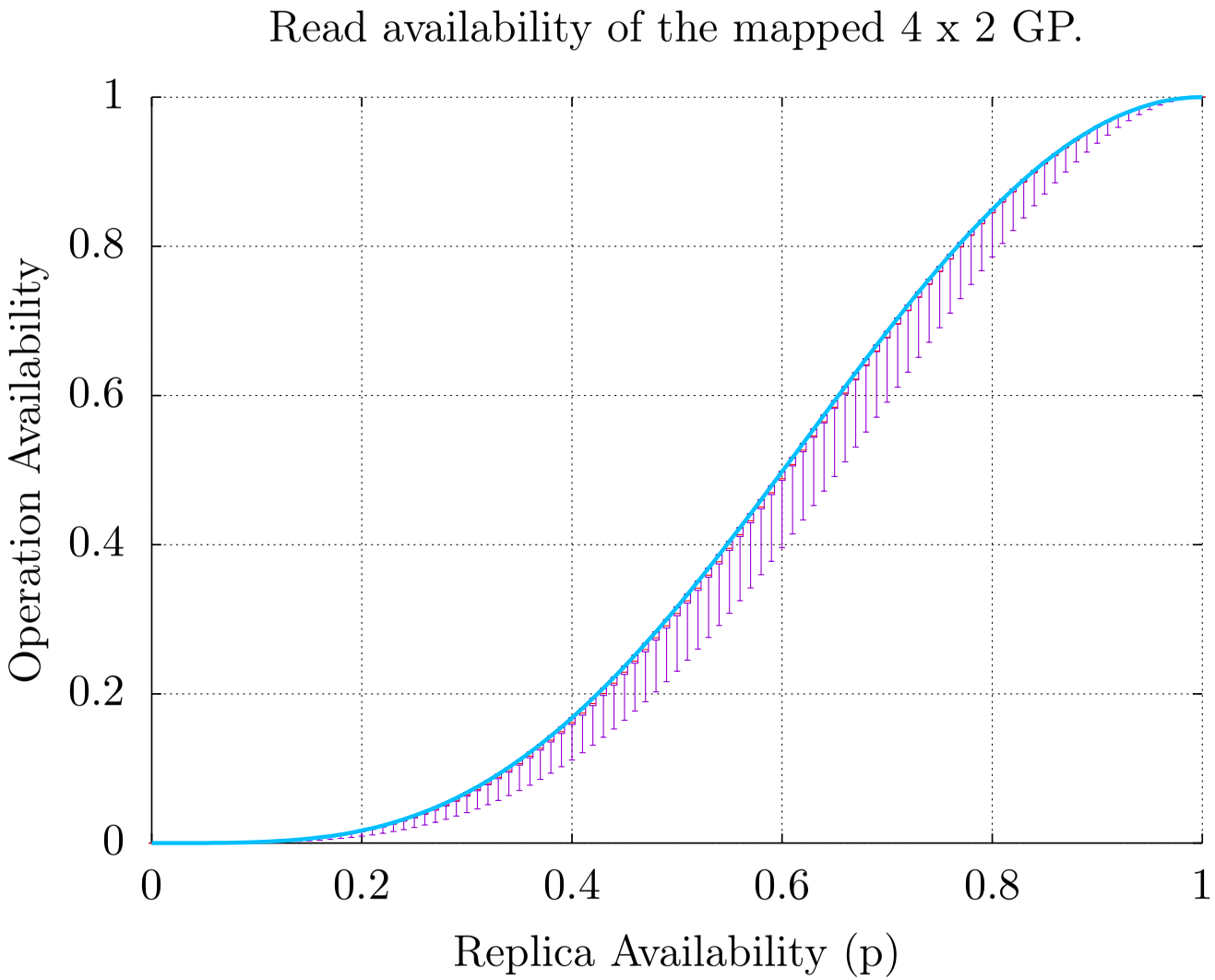


Figure 5.25: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ of the GP with 4×2 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_r(p)$ of the GP with 4×2 replicas in its unmapped state.

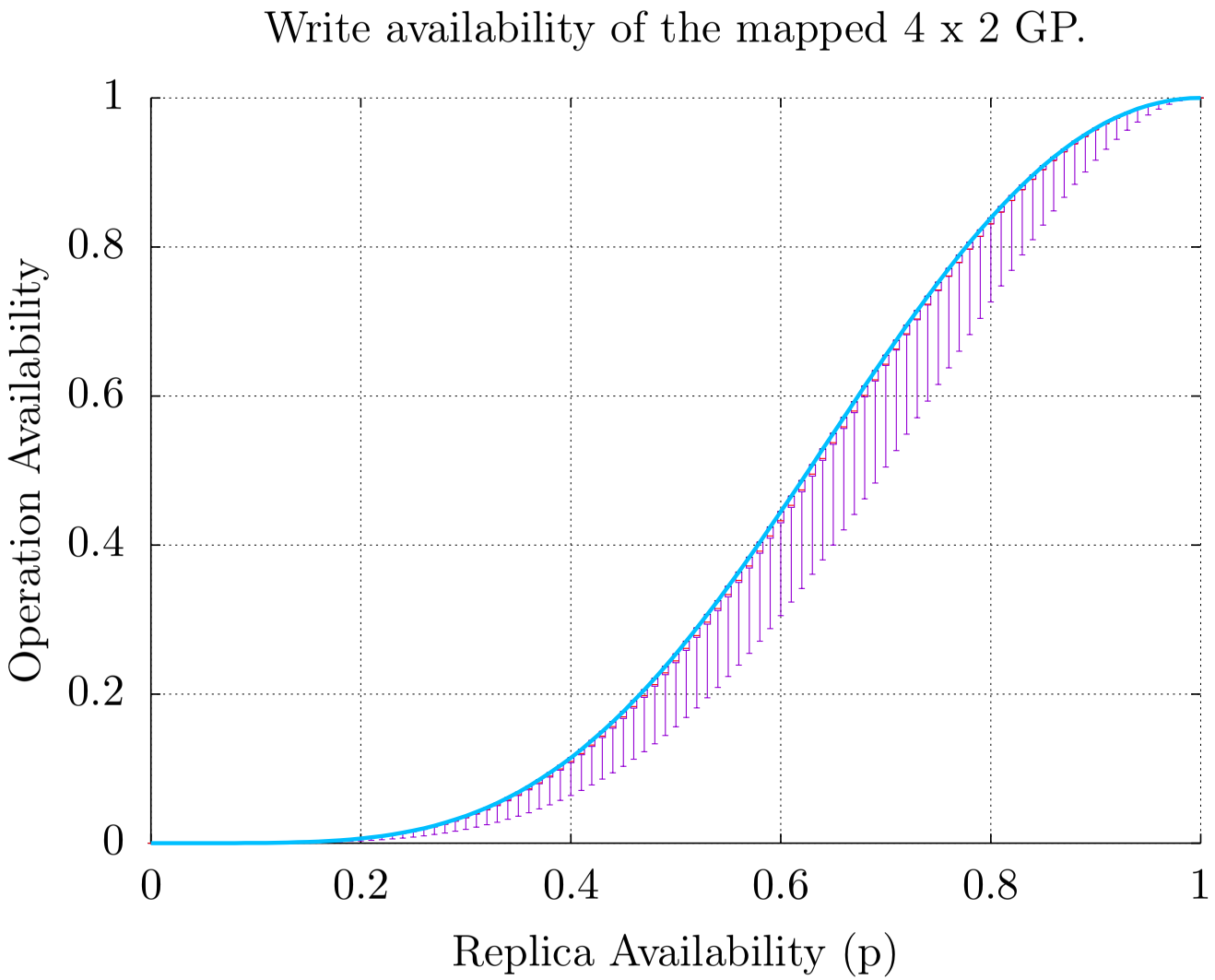


Figure 5.26: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_w(p)$ of the GP with 4×2 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_w(p)$ of the GP with 4×2 replicas in its unmapped state.

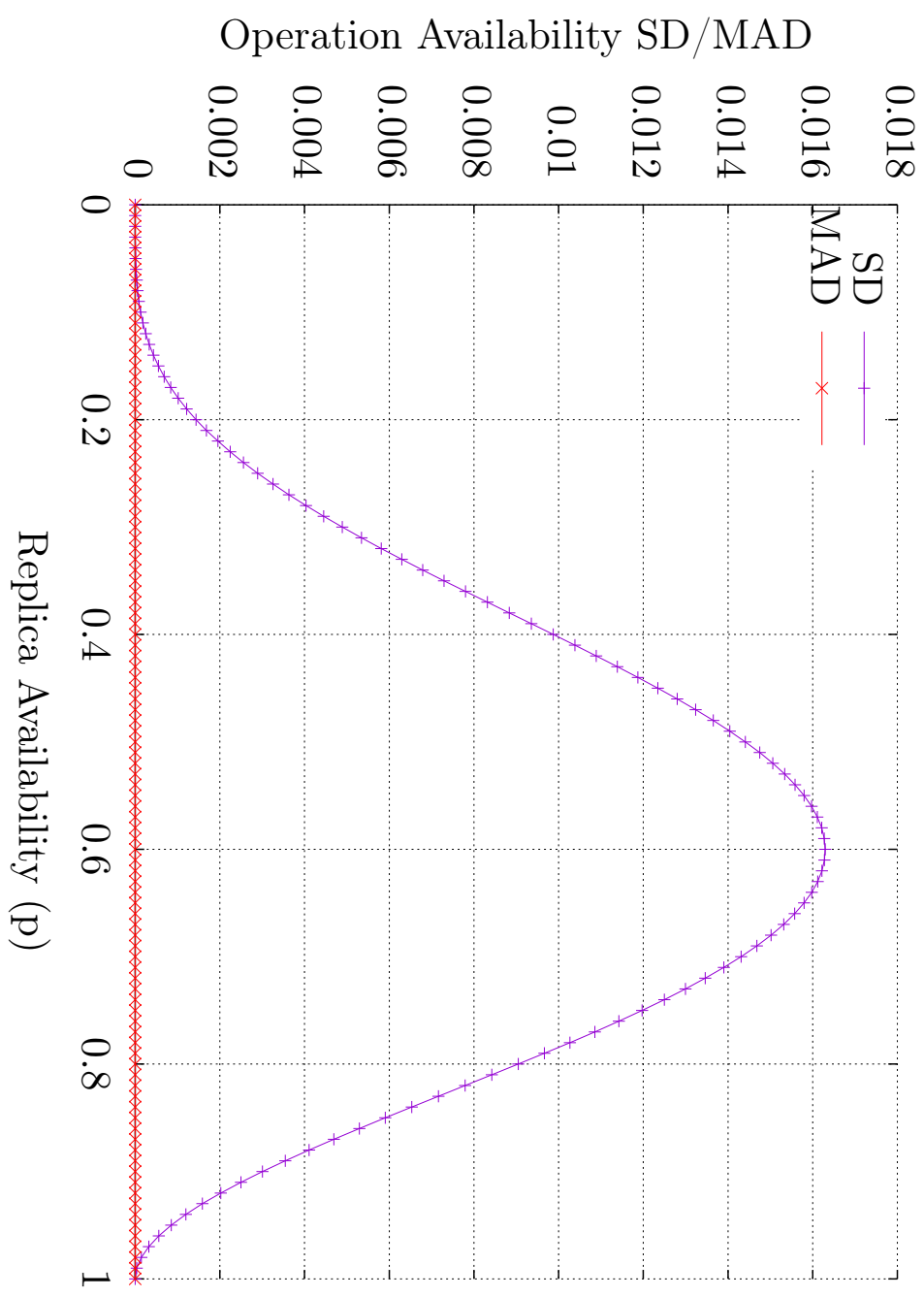


Figure 5.27: The minimal SD and MAD of the $a_r(p)$ of the GP with 4×2 replicas mapped to 255 simple, non-isomorphic GSs.

Figure 5.28: The minimal SD and MAD of the $a_w(p)$ of the GP with 4×2 replicas mapped to 255 simple, non-isomorphic GSs.

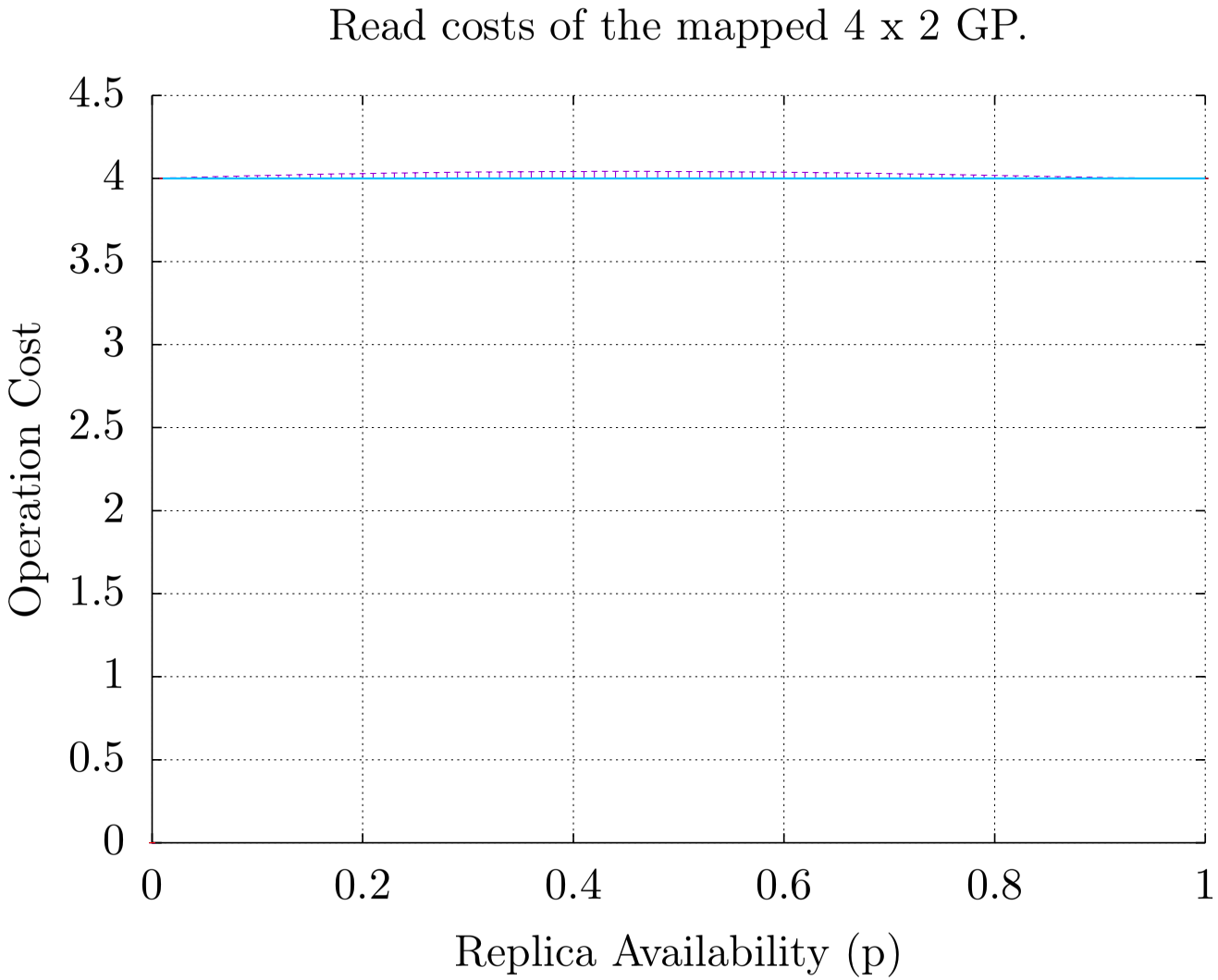


Figure 5.29: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ of the GP with 4×2 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_r(p)$ of the GP with 4×2 replicas in its unmapped state.

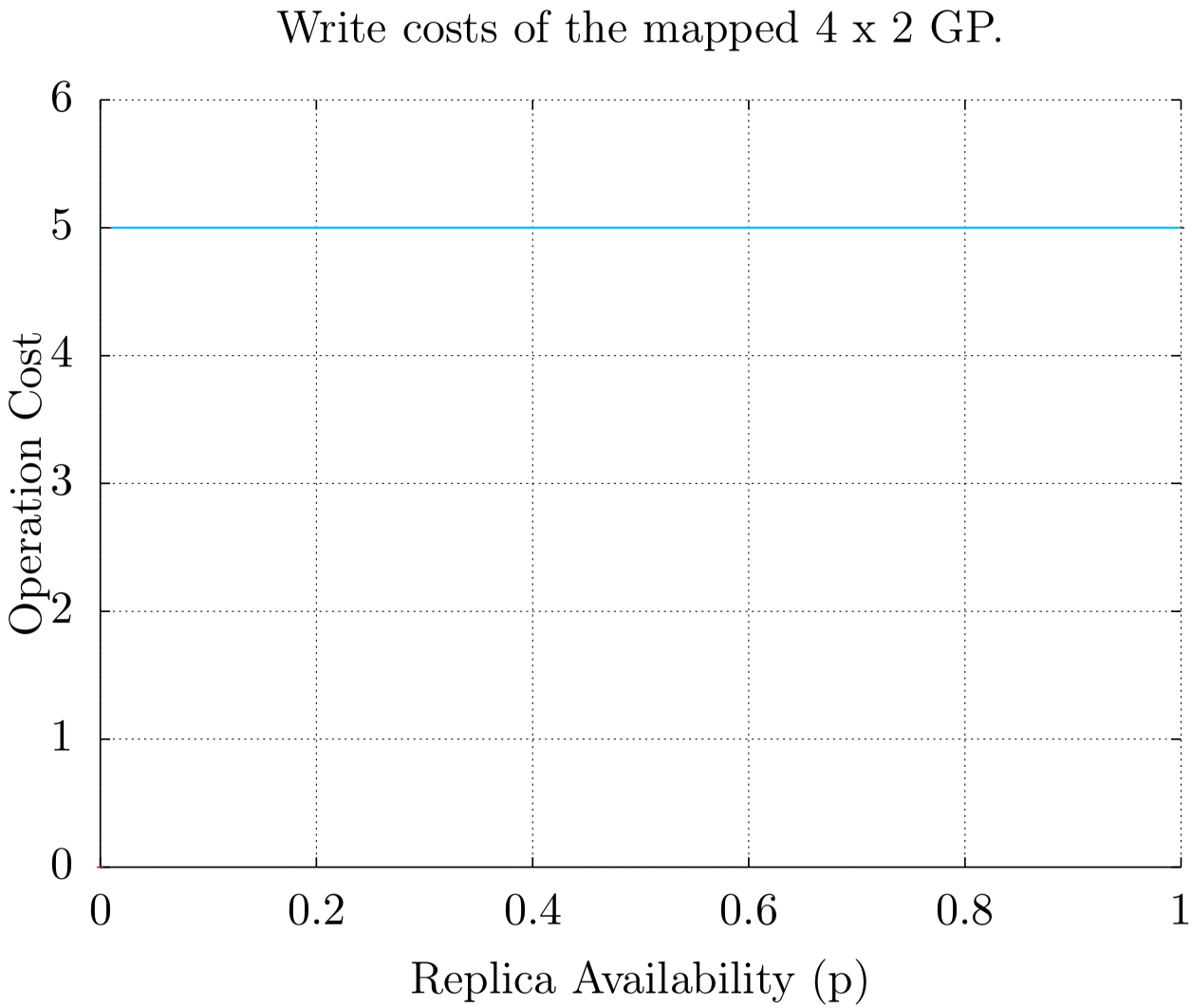


Figure 5.30: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_w(p)$ of the GP with 4×2 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_w(p)$ of the GP with 4×2 replicas in its unmapped state.

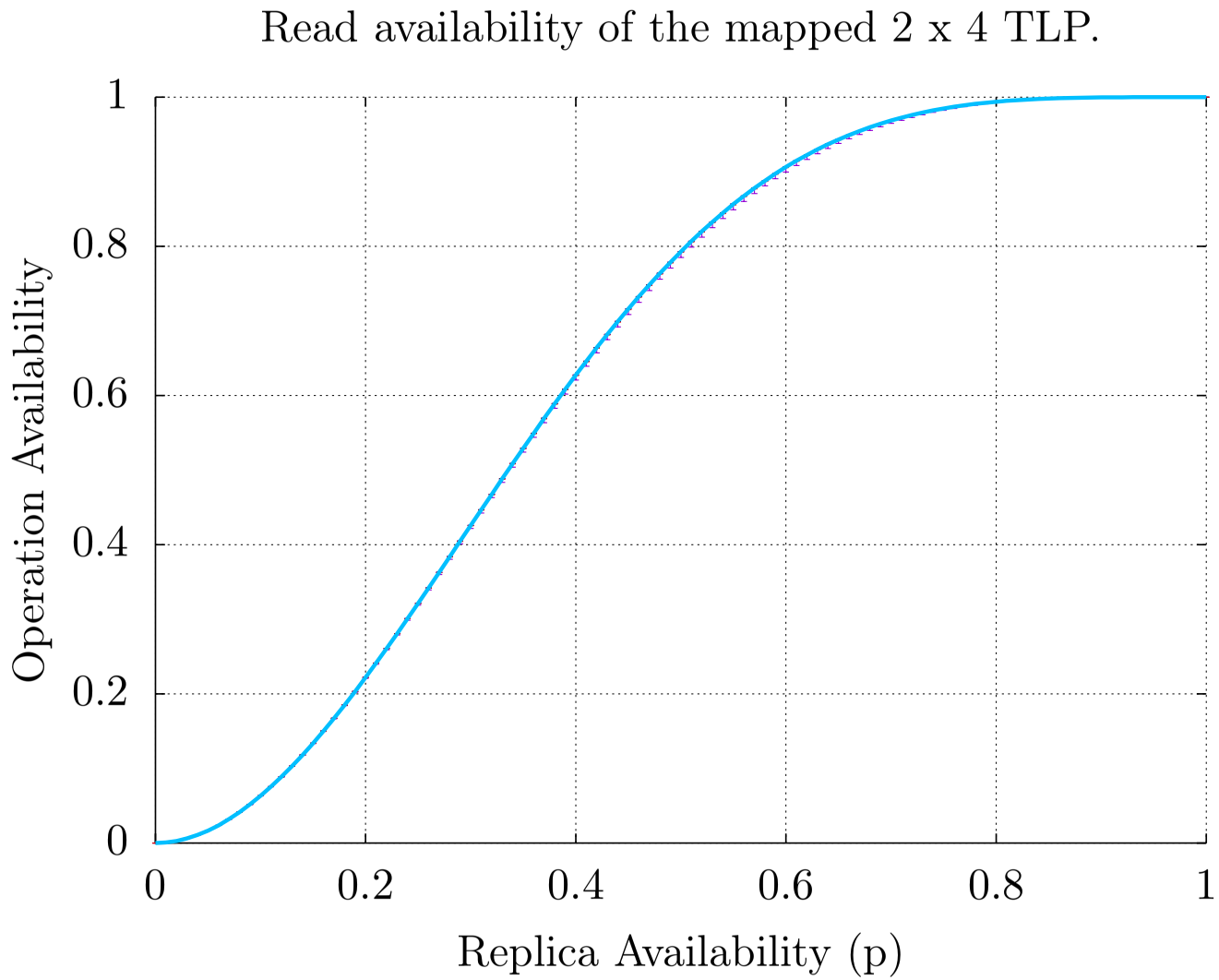


Figure 5.31: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ of the TLP with 2×4 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_r(p)$ of the TLP with 2×4 replicas in its unmapped state.

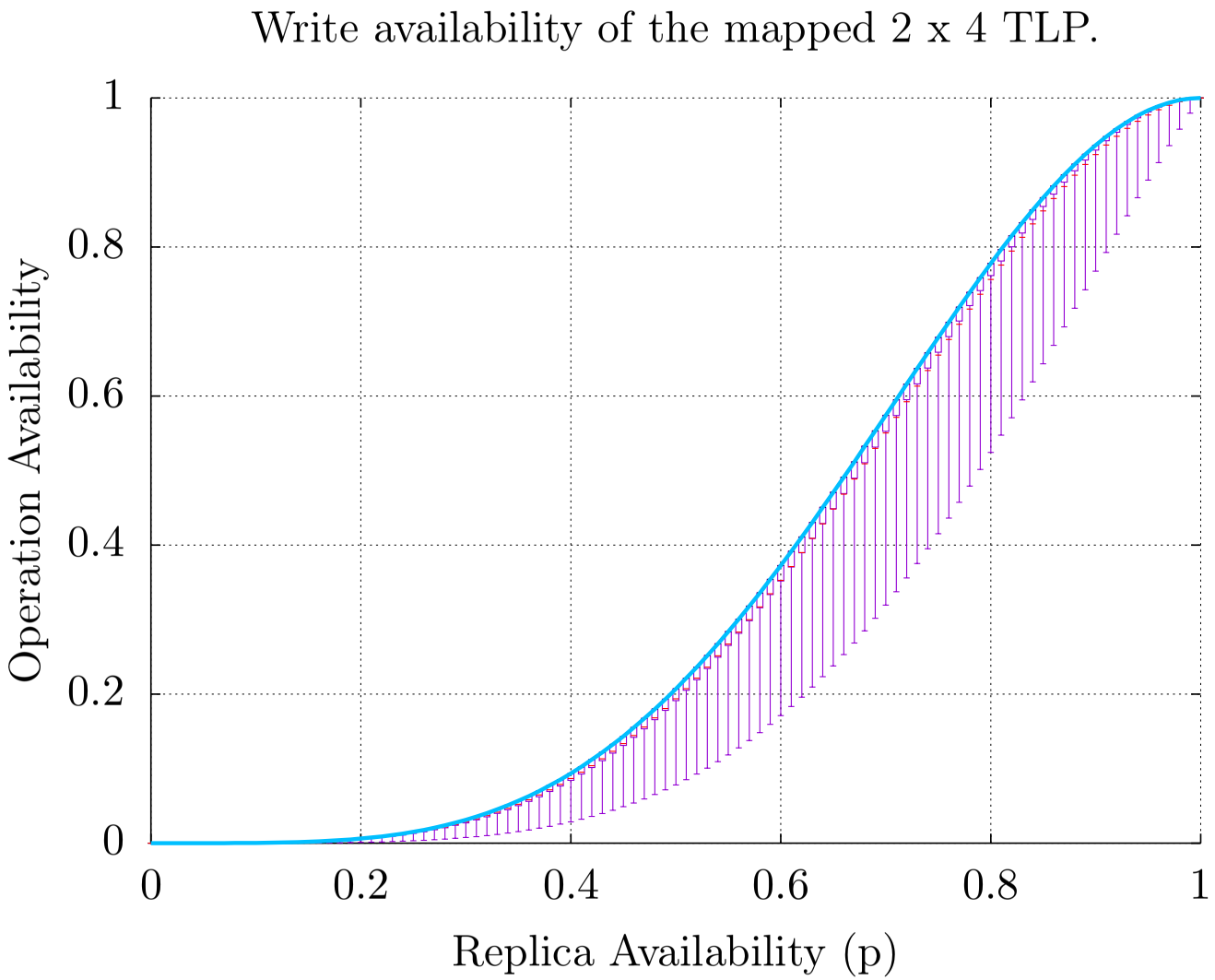


Figure 5.32: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_w(p)$ of the TLP with 2×4 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_w(p)$ of the TLP with 2×4 replicas in its unmapped state.

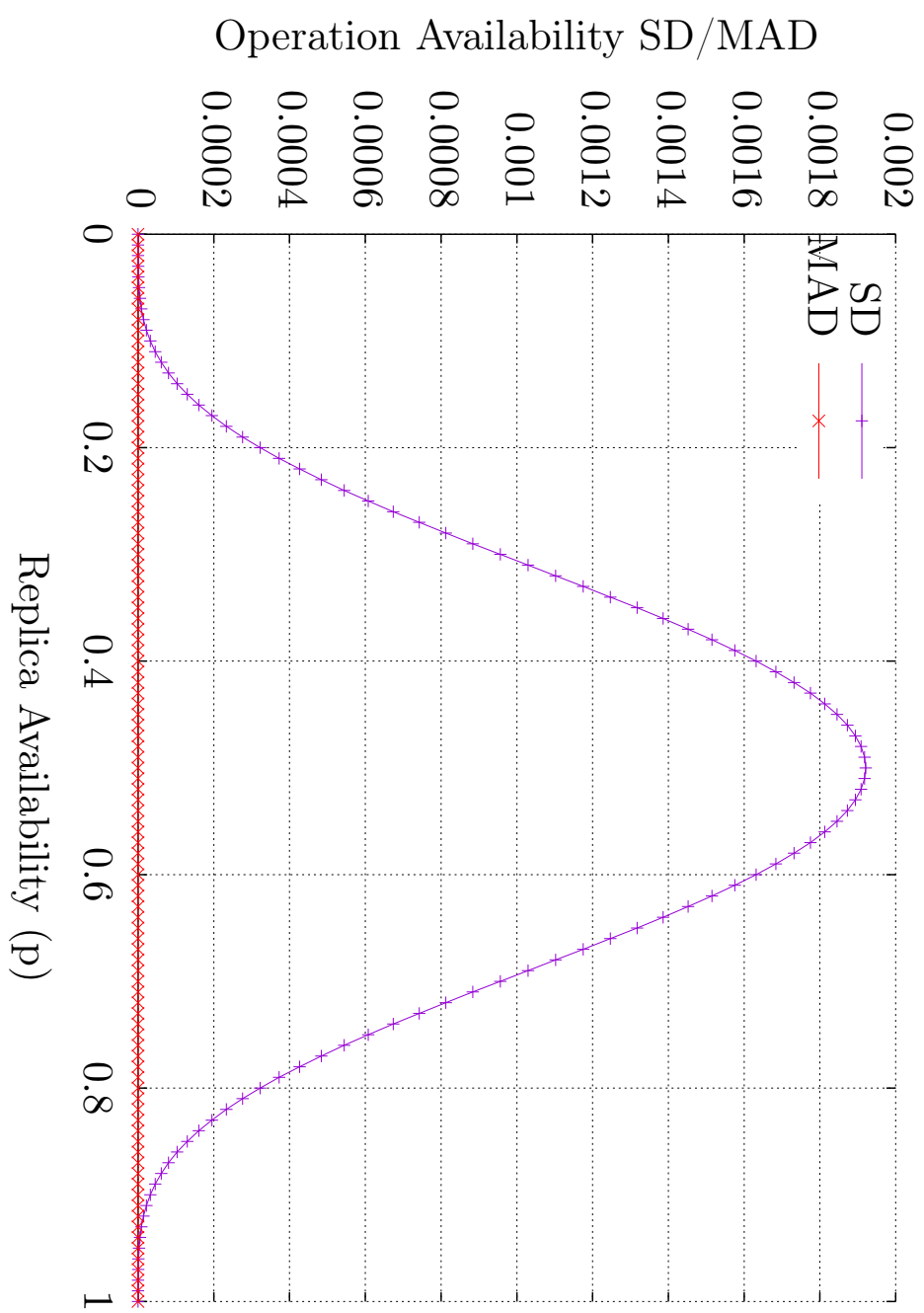


Figure 5.33: The minimal SD and MAD of the $a_r(p)$ of the TLP with 2×4 replicas mapped to 255 simple, non-isomorphic GSs.

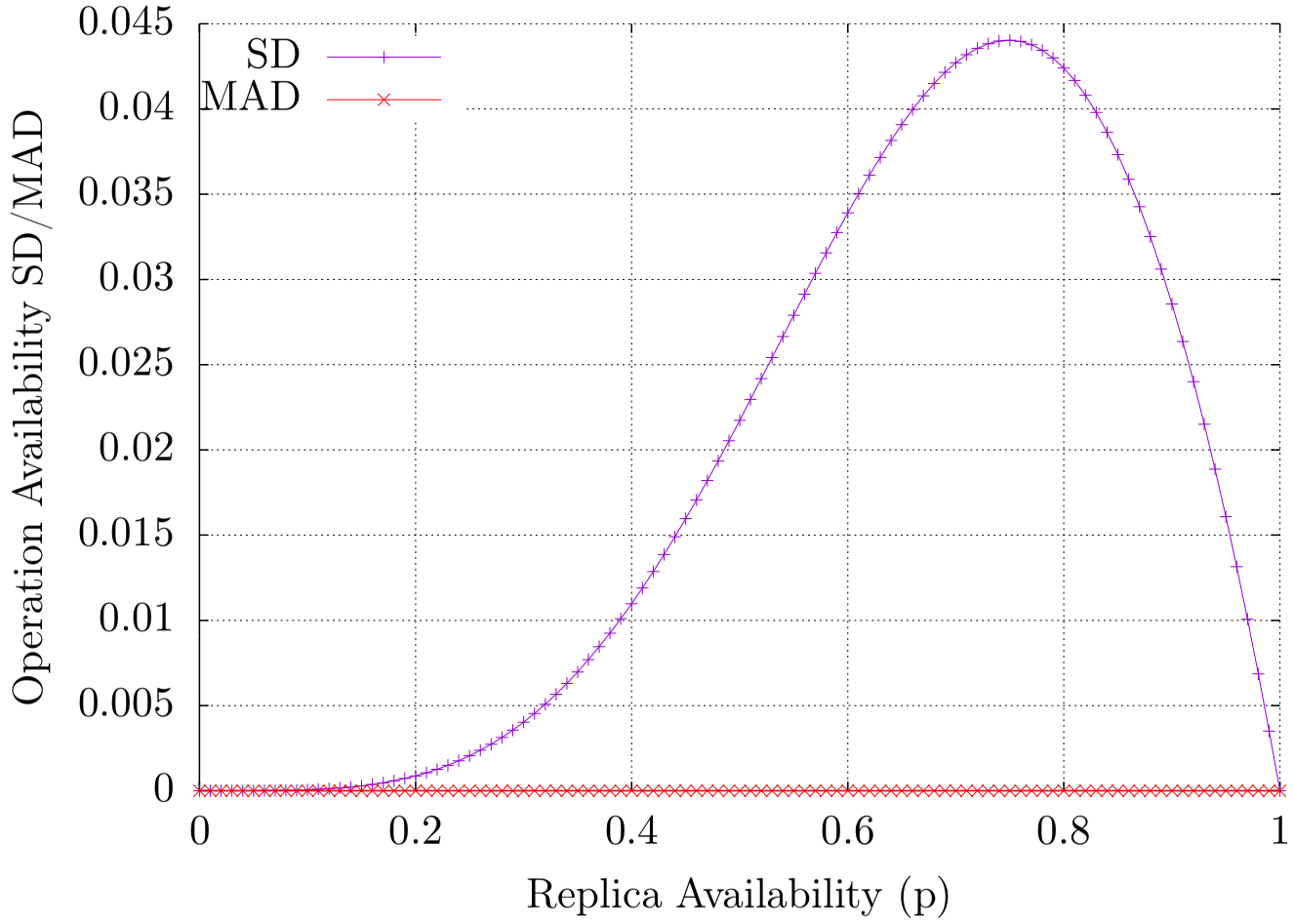


Figure 5.34: The minimal SD and MAD of the $a_w(p)$ of the TLP with 2×4 replicas mapped to 255 simple, non-isomorphic GSs.

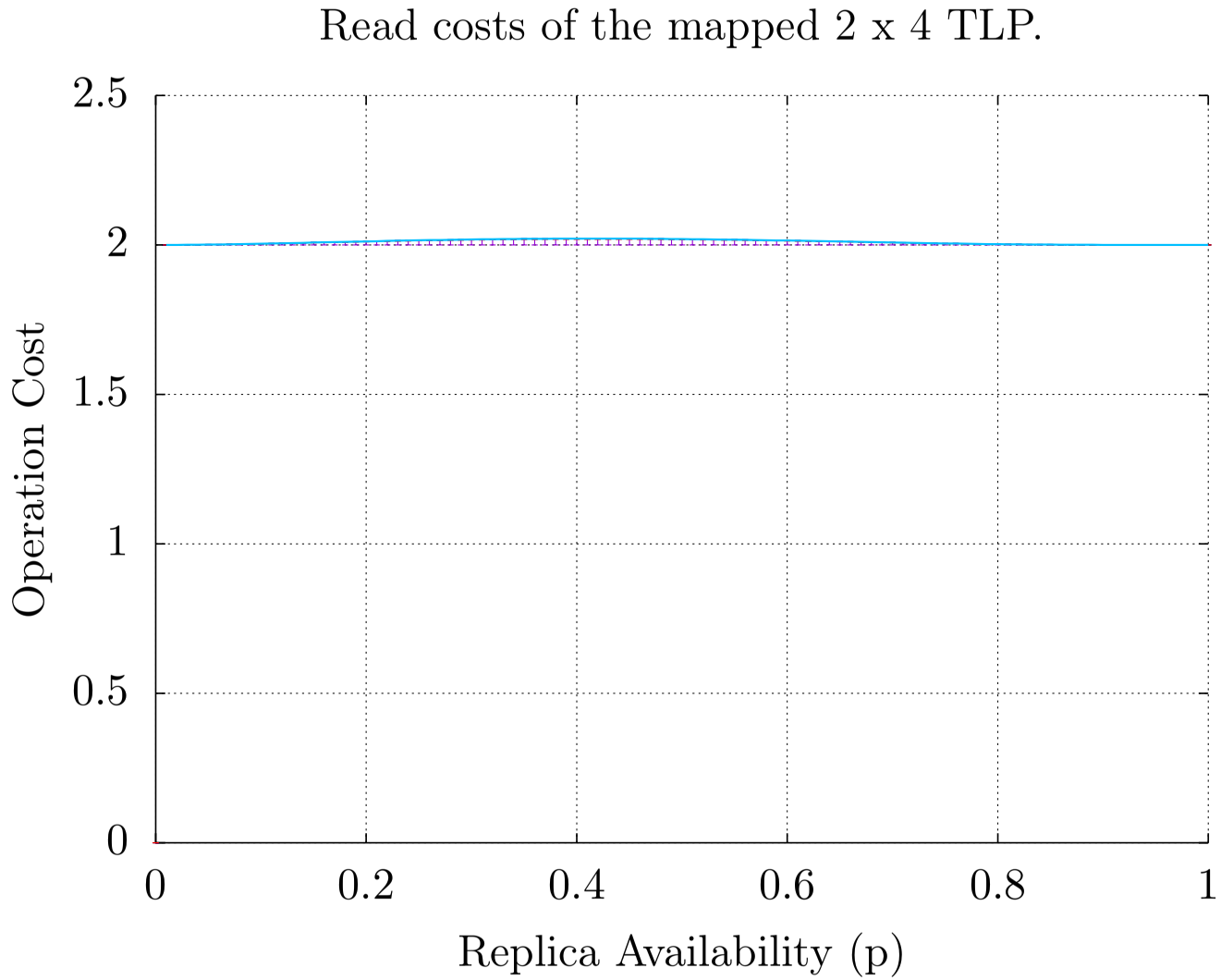


Figure 5.35: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ of the TLP with 2×4 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_r(p)$ of the TLP with 2×4 replicas in its unmapped state.

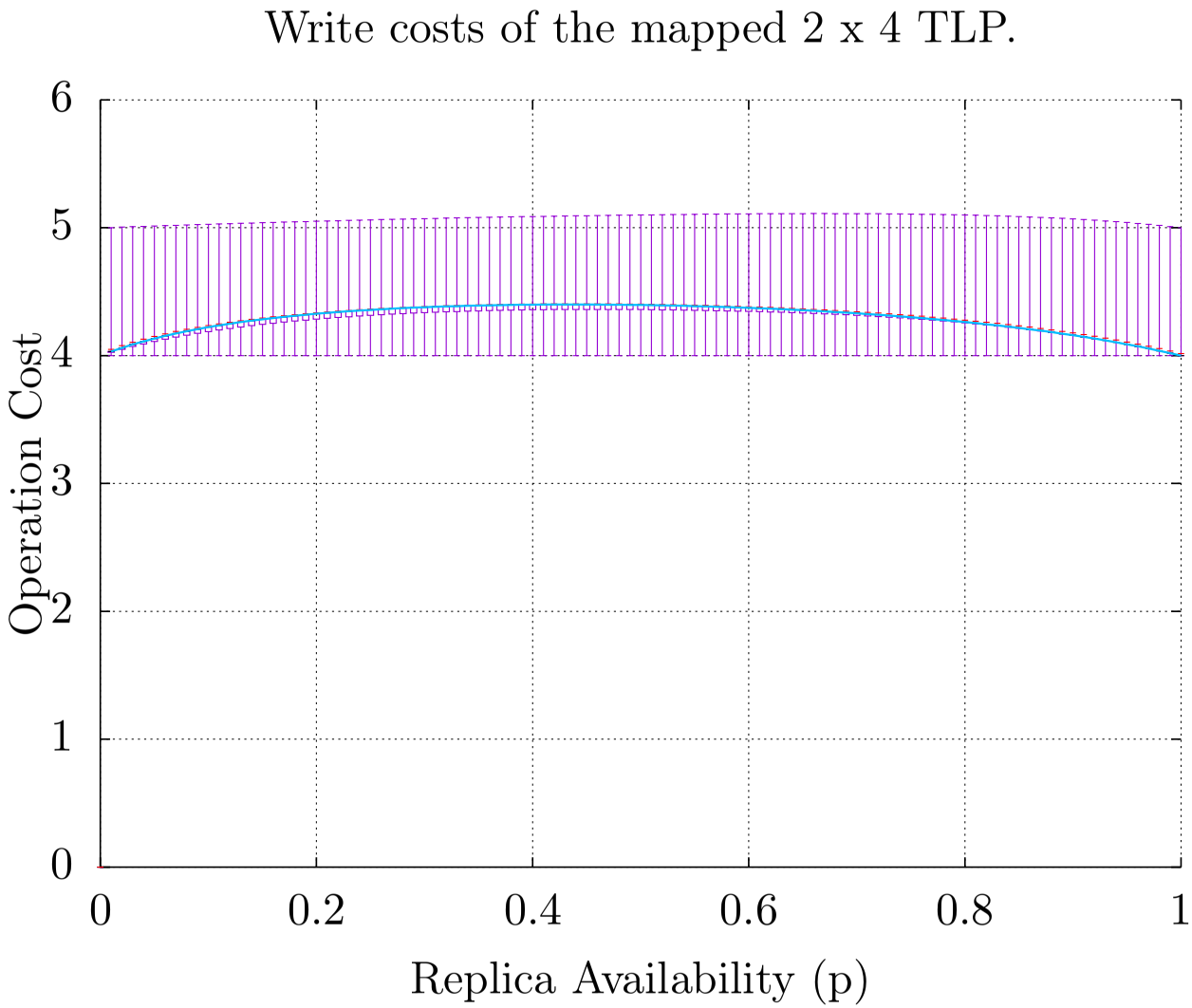


Figure 5.36: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_w(p)$ of the TLP with 2×4 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_w(p)$ of the TLP with 2×4 replicas in its unmapped state.

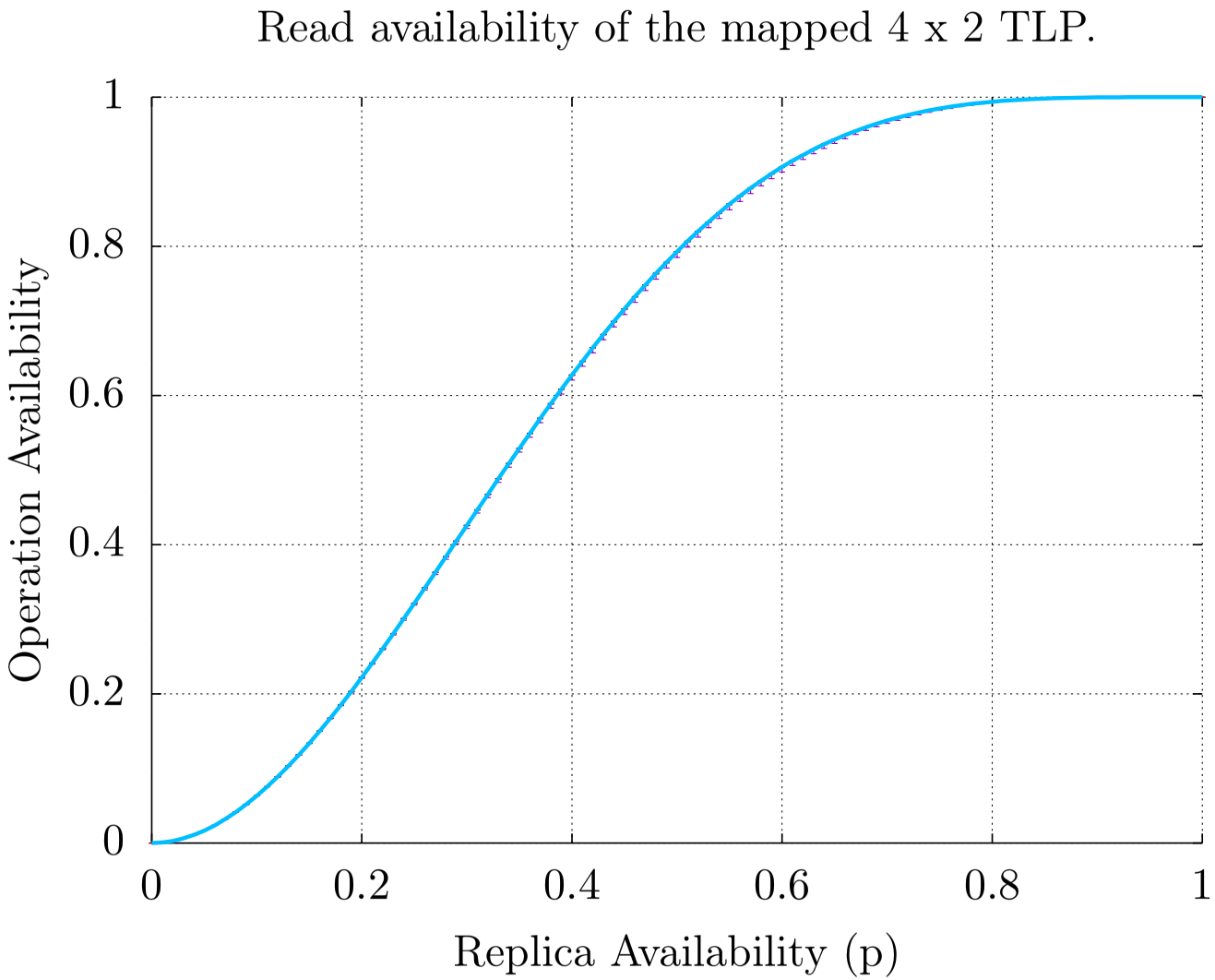


Figure 5.37: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ of the TLP with 4×2 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_r(p)$ of the TLP with 4×2 replicas in its unmapped state.

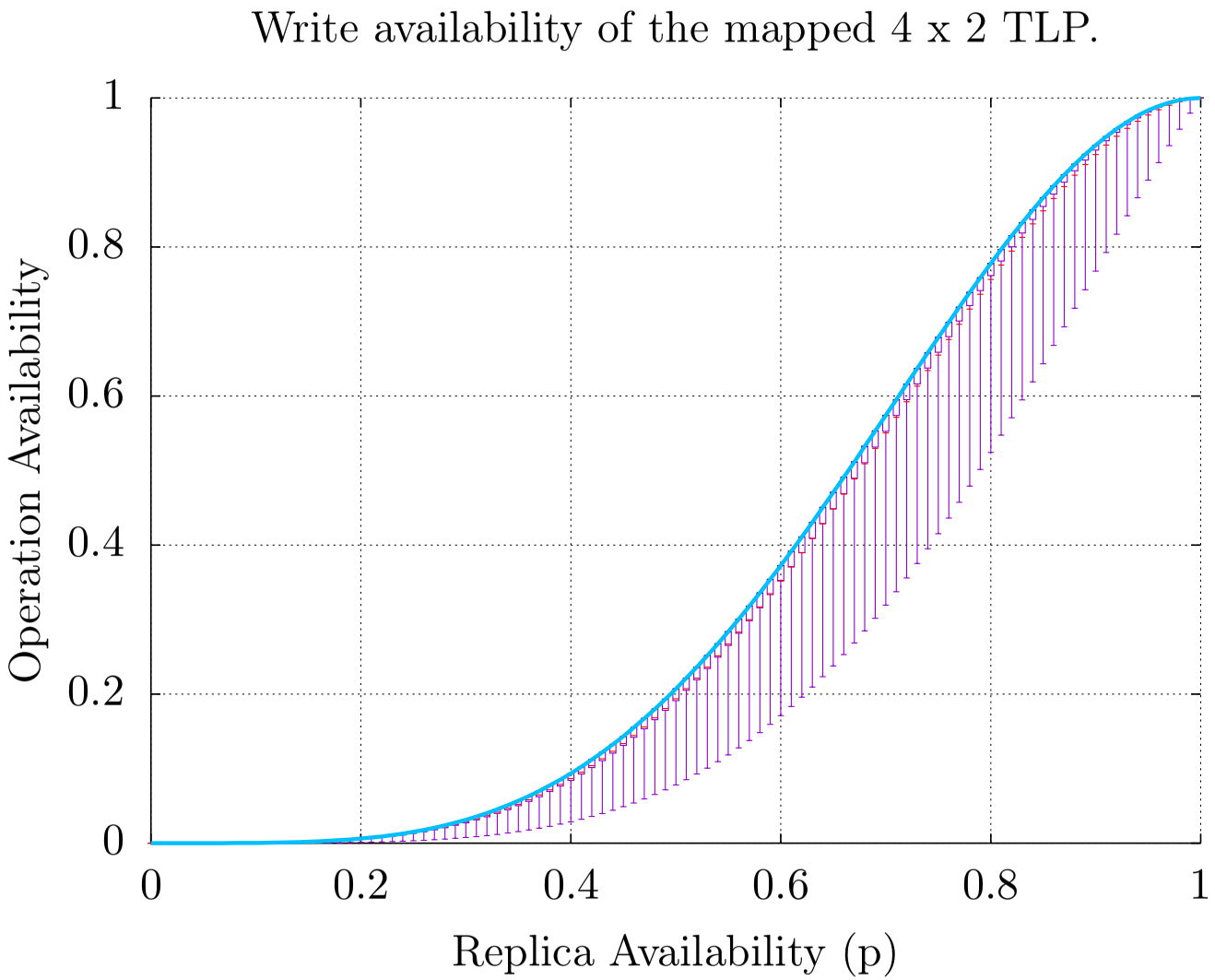


Figure 5.38: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_w(p)$ of the TLP with 4×2 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_w(p)$ of the TLP with 4×2 replicas in its unmapped state.

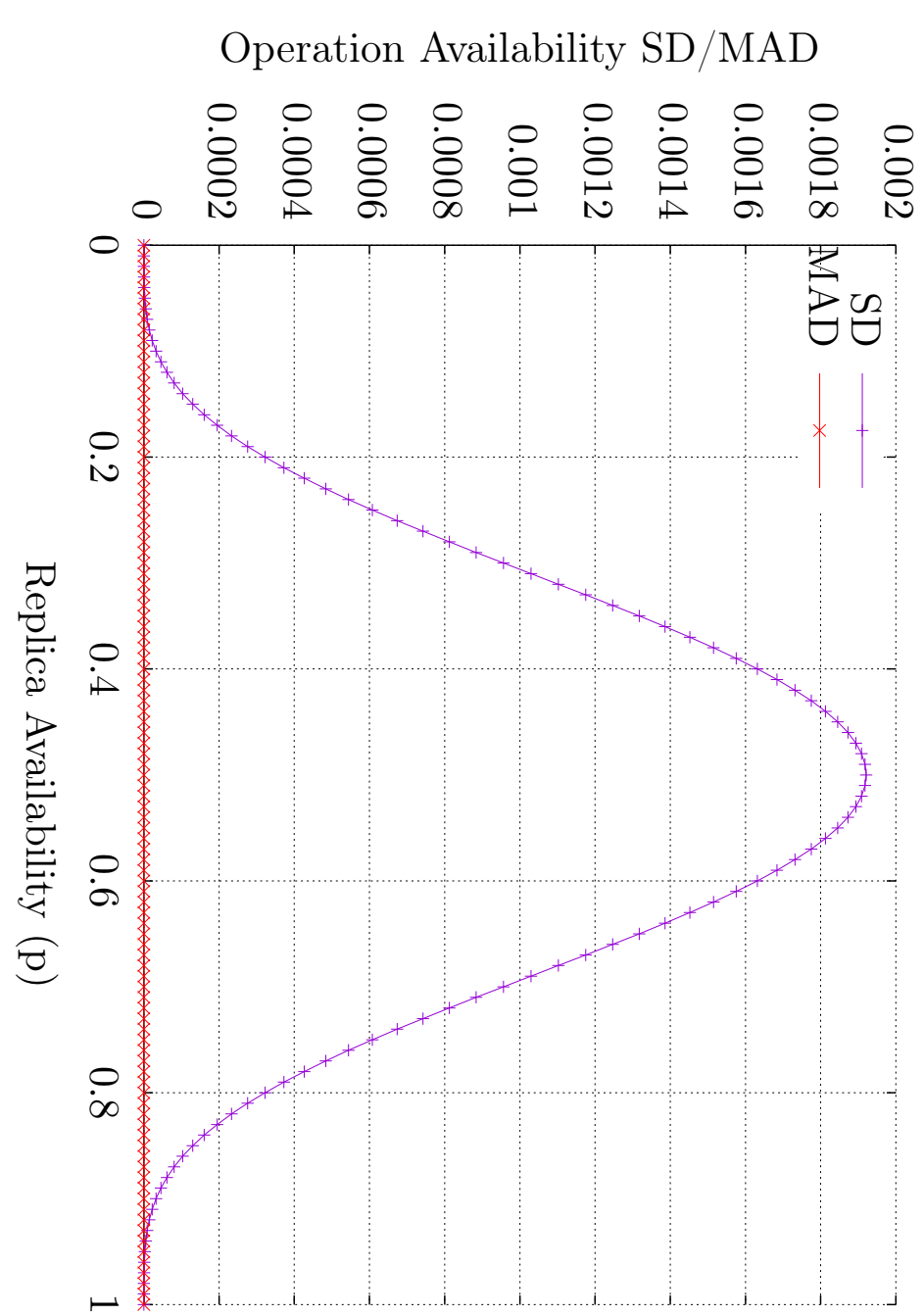


Figure 5.39: The minimal SD and MAD of the $a_r(p)$ of the TLP with 4×2 replicas mapped to 255 simple, non-isomorphic GSs.

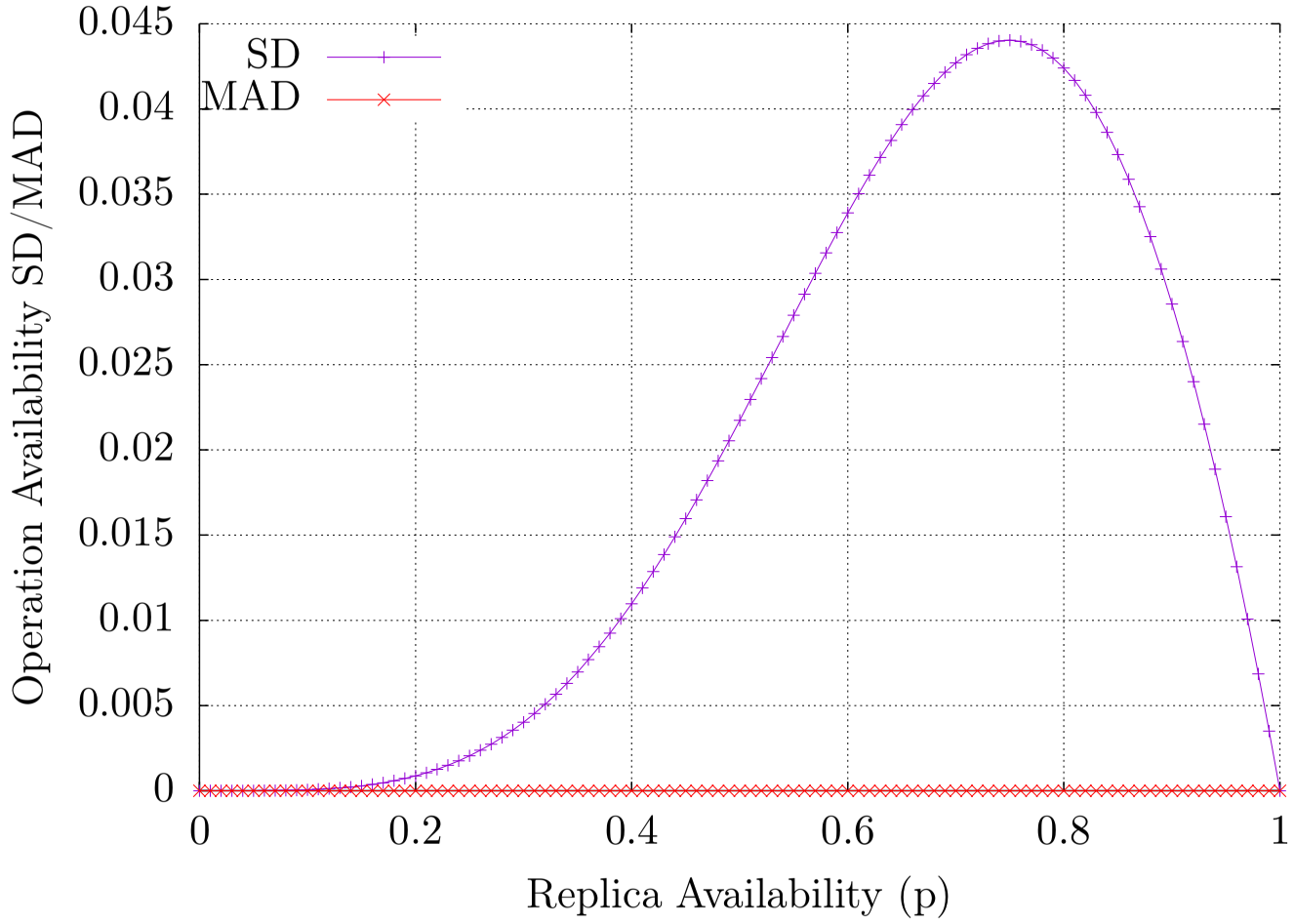


Figure 5.40: The minimal SD and MAD of the $a_w(p)$ of the TLP with 4×2 replicas mapped to 255 simple, non-isomorphic GSs.

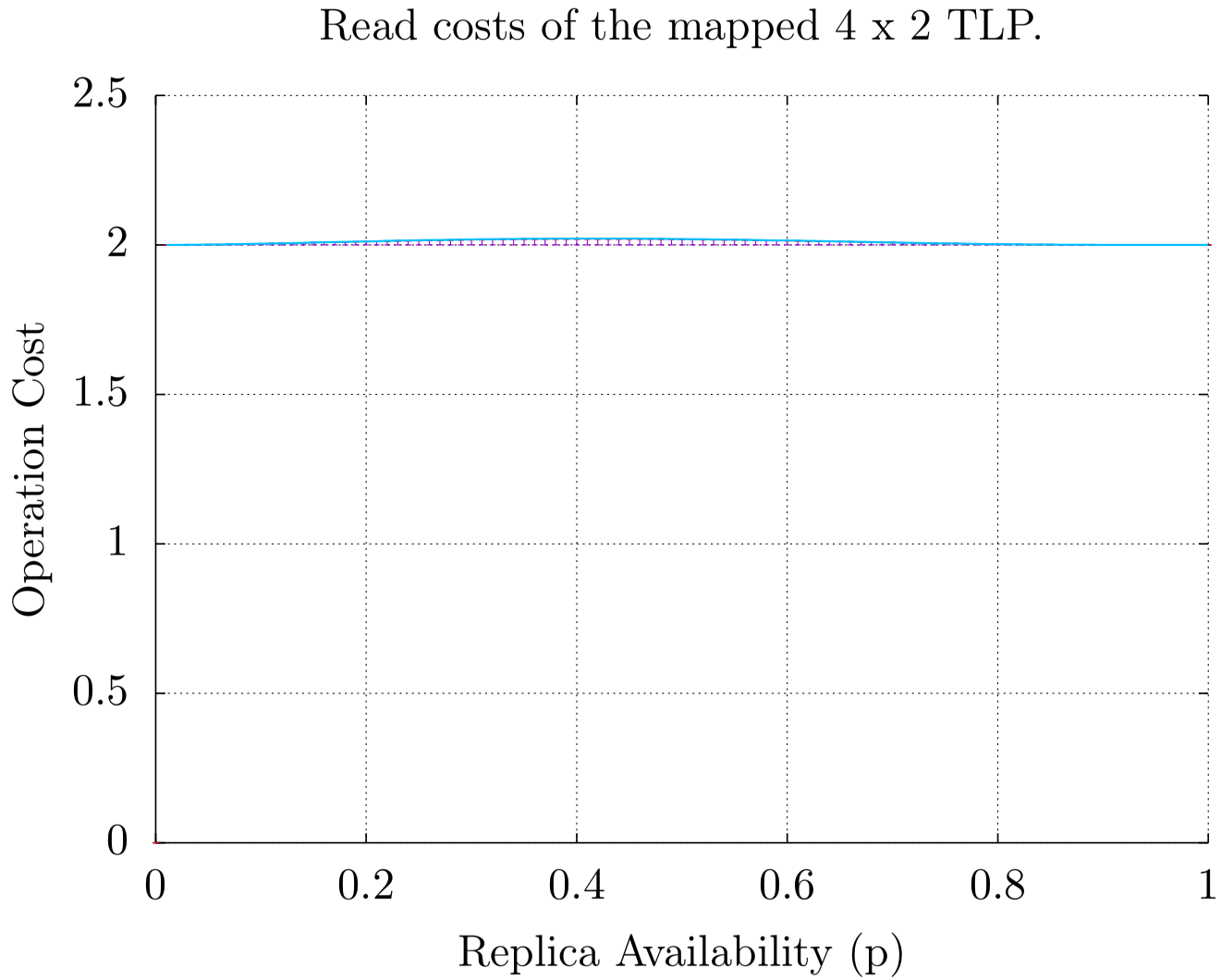


Figure 5.41: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ of the TLP with 4×2 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_r(p)$ of the TLP with 4×2 replicas in its unmapped state.

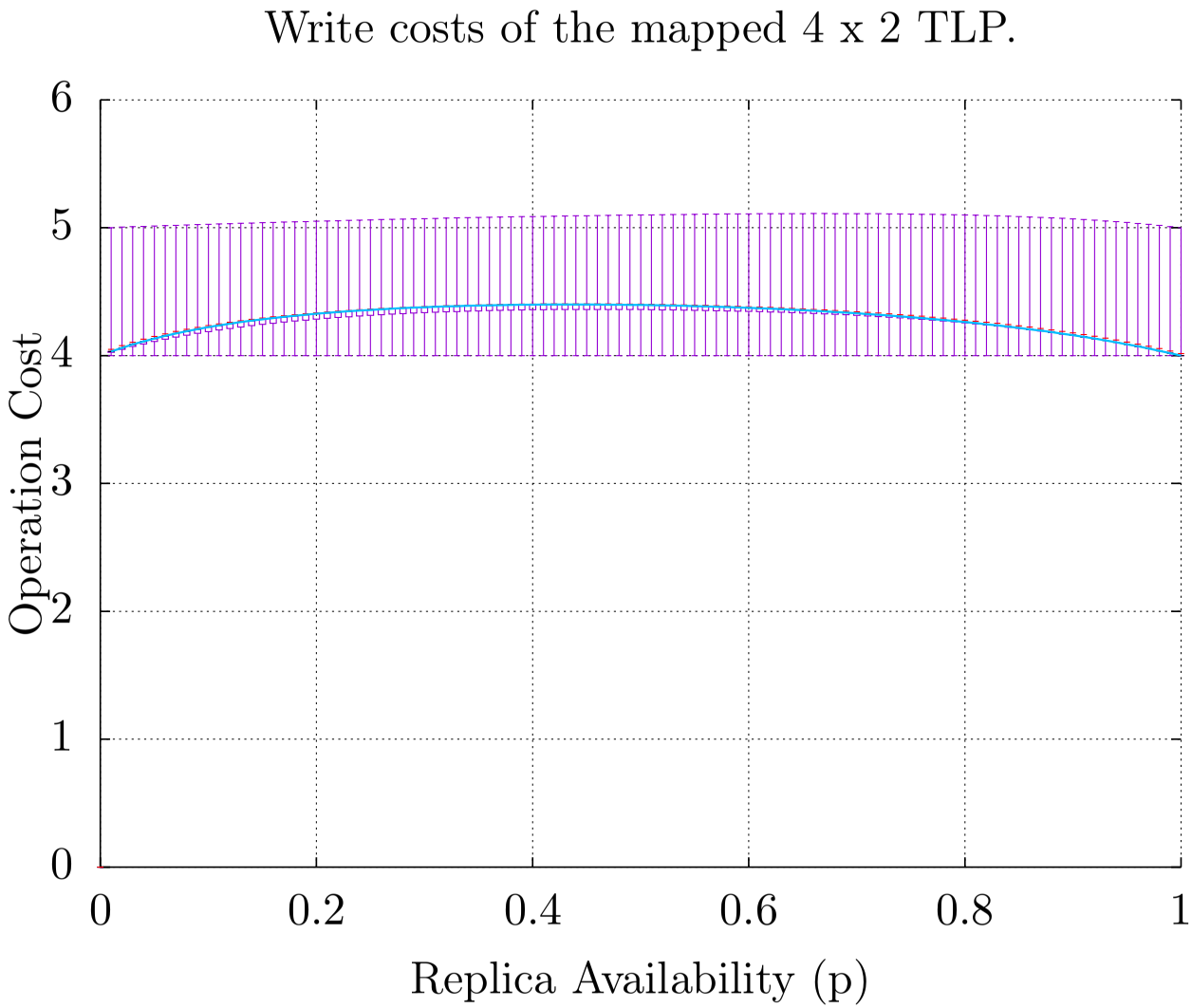


Figure 5.42: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_w(p)$ of the TLP with 4×2 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_w(p)$ of the TLP with 4×2 replicas in its unmapped state.

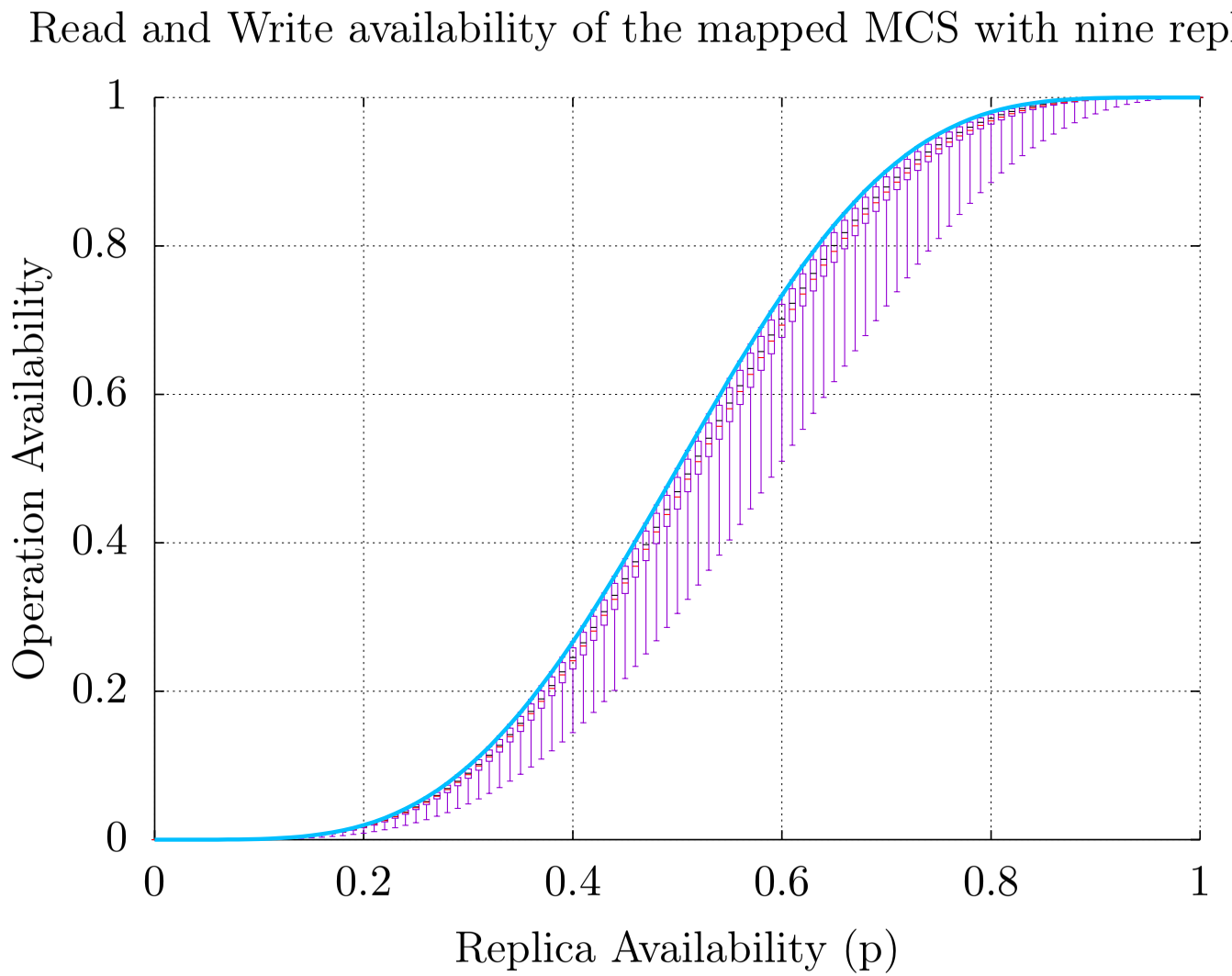


Figure 5.43: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ of the MCS with nine replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_r(p)$ of the MCS with nine replicas in its unmapped state. All noted values are five, and therefore overlap in the plot.

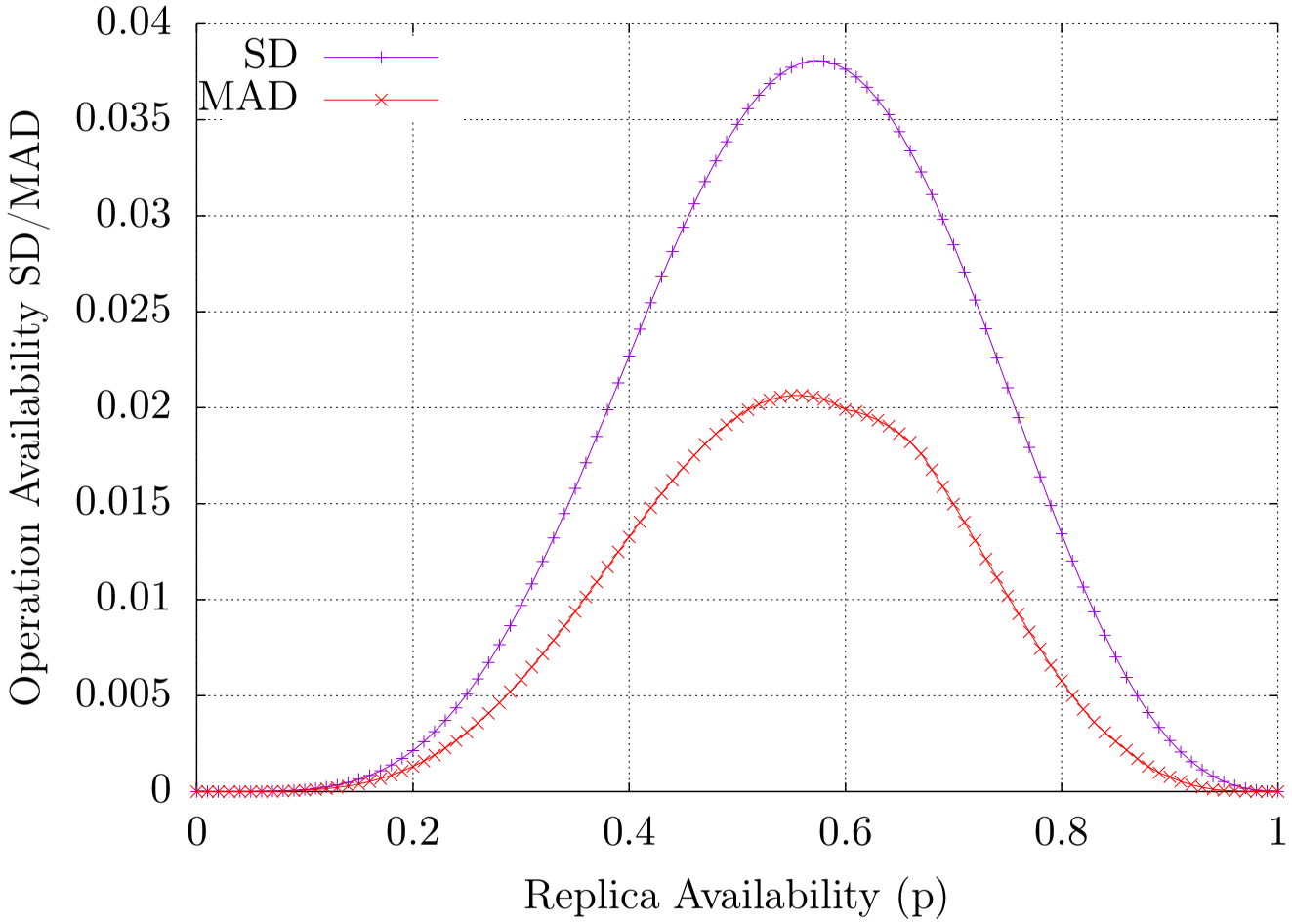


Figure 5.44: The minimal SD and MAD of the $a_r(p)$ of the MCS with nine replicas mapped to 255 simple, non-isomorphic GSs.

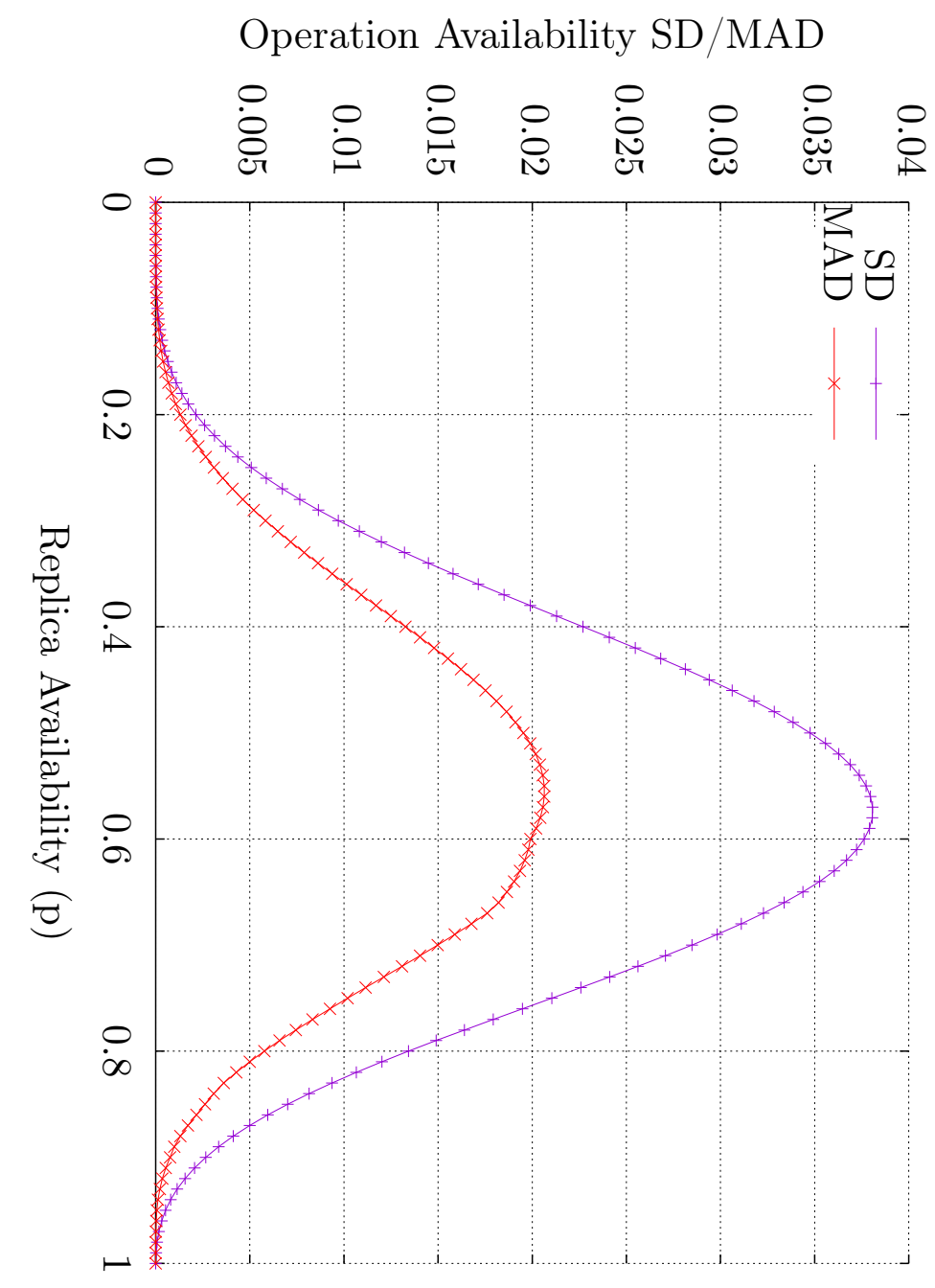


Figure 5.45: The minimal SD and MAD of the $a_w(p)$ of the MCS with nine replicas mapped to 255 simple, non-isomorphic GSs.

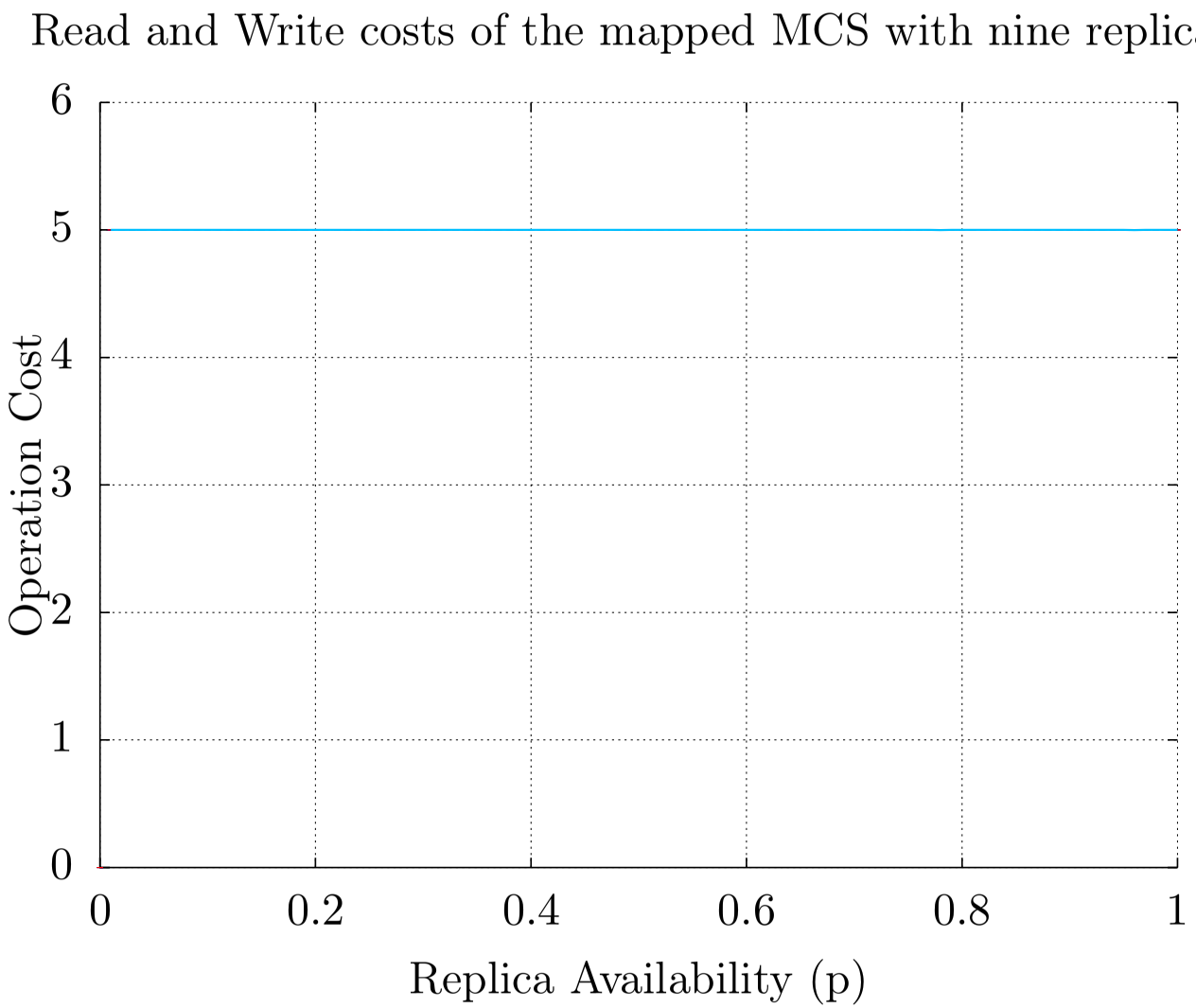


Figure 5.46: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ of the MCS with nine replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_r(p)$ of the MCS with nine replicas in its unmapped state.

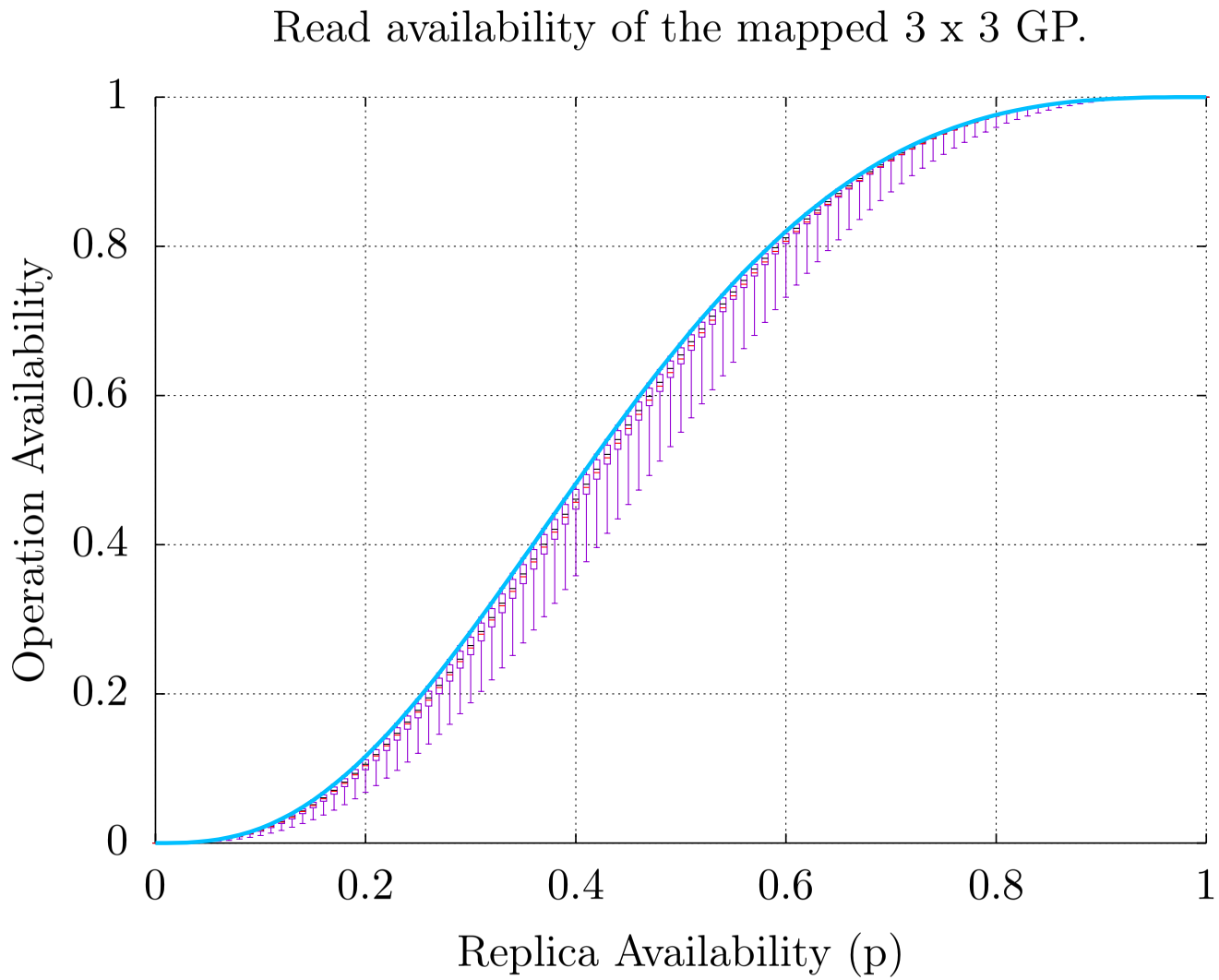


Figure 5.47: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ of the GP with 3×3 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_r(p)$ of the GP with 3×3 replicas in its unmapped state.

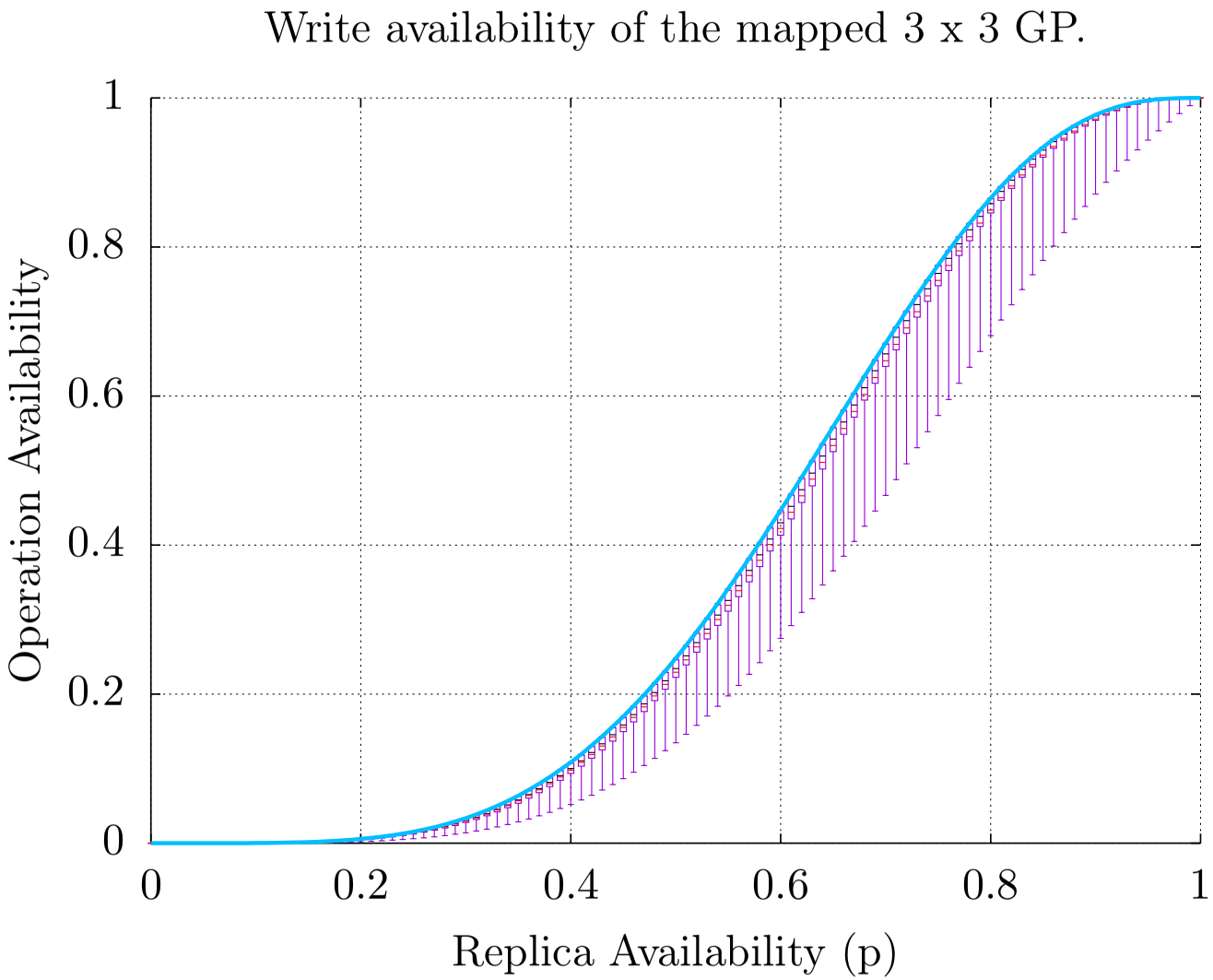


Figure 5.48: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_w(p)$ of the GP with 3×3 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_w(p)$ of the GP with 3×3 replicas in its unmapped state.

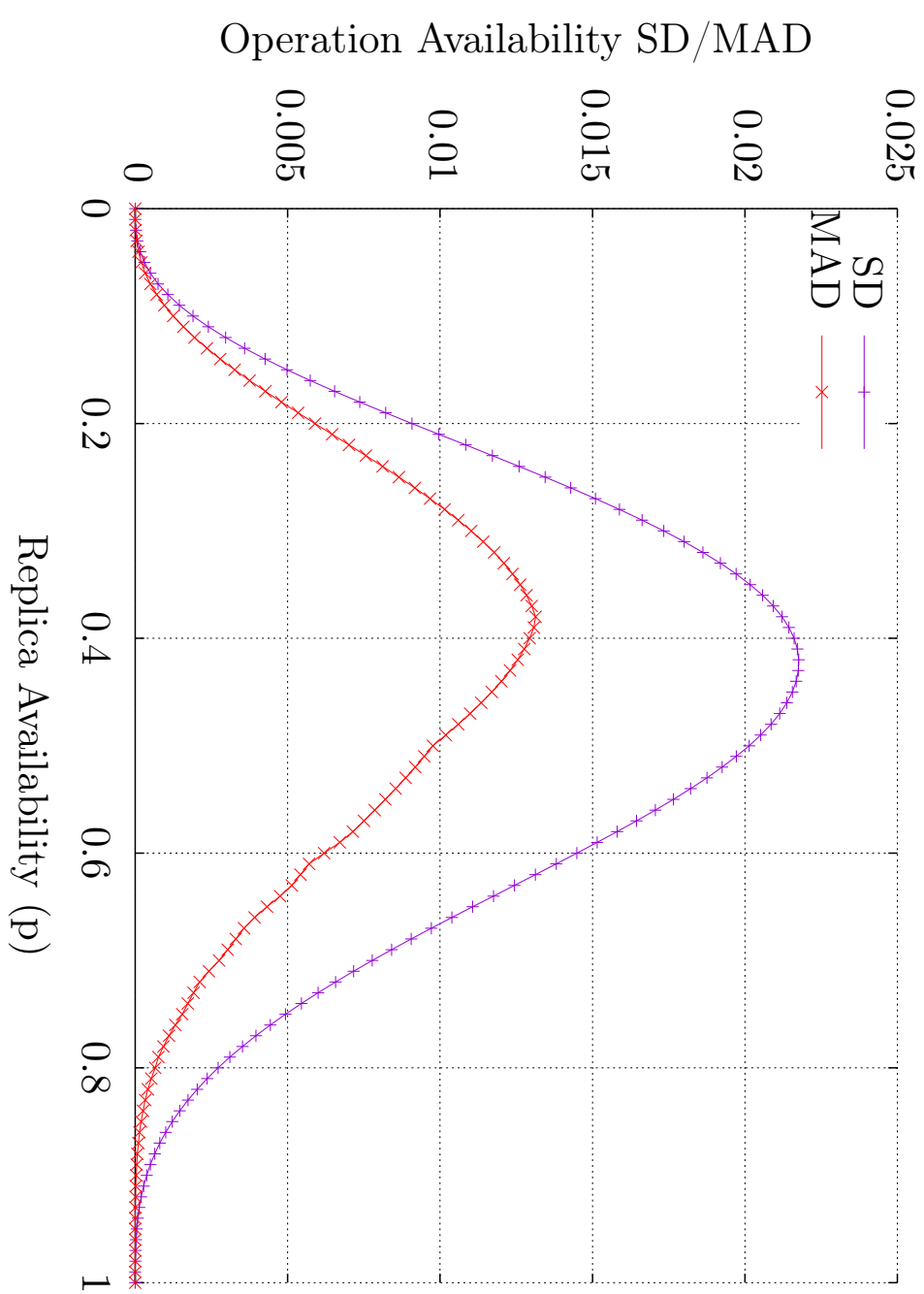


Figure 5.49: The minimal SD and MAD of the $a_r(p)$ of the GP with 3×3 replicas mapped to 255 simple, non-isomorphic GSs.

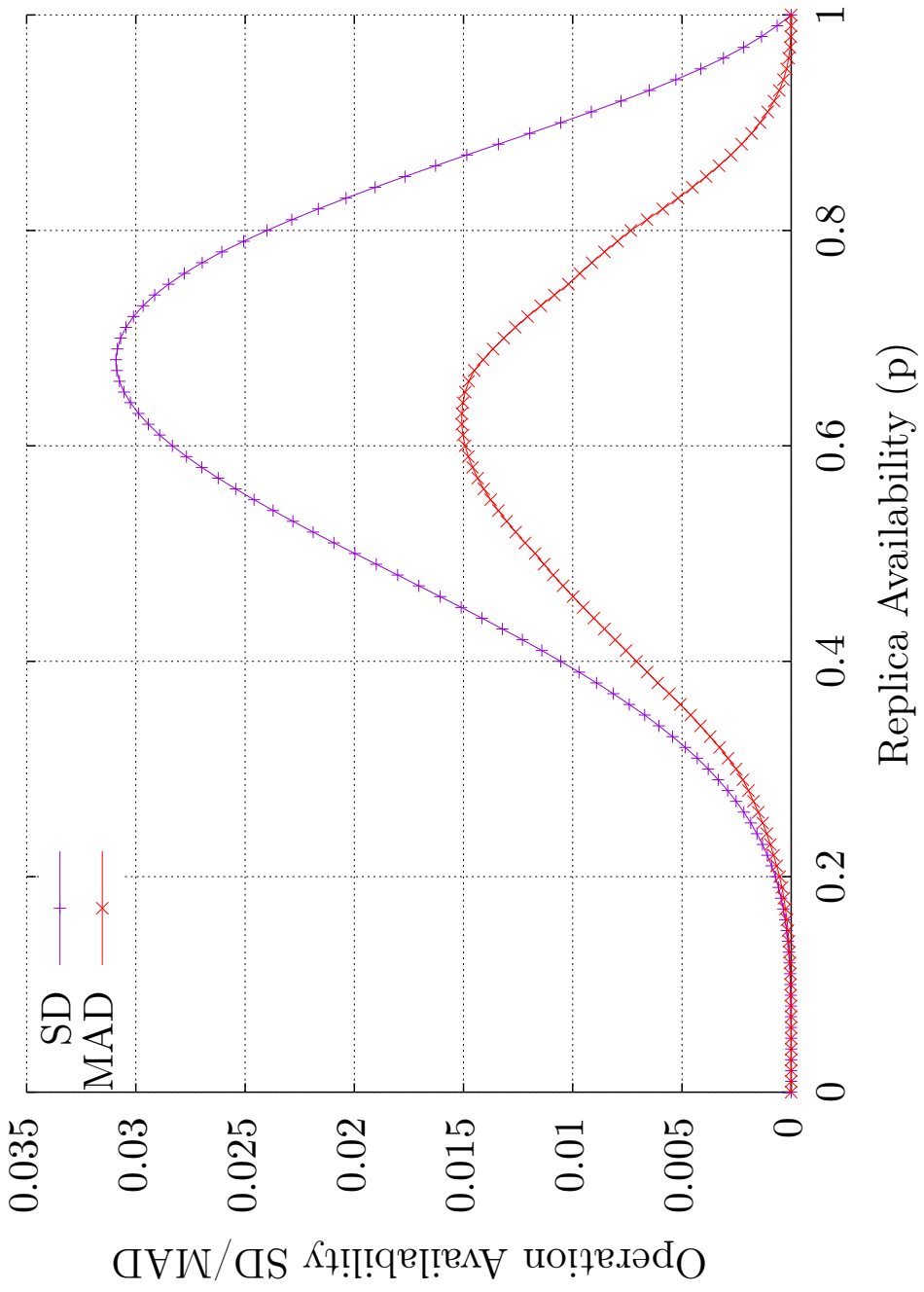


Figure 5.50: The minimal SD and MAD of the $a_w(p)$ of the GP with 3×3 replicas mapped to 255 simple, non-isomorphic GSs.

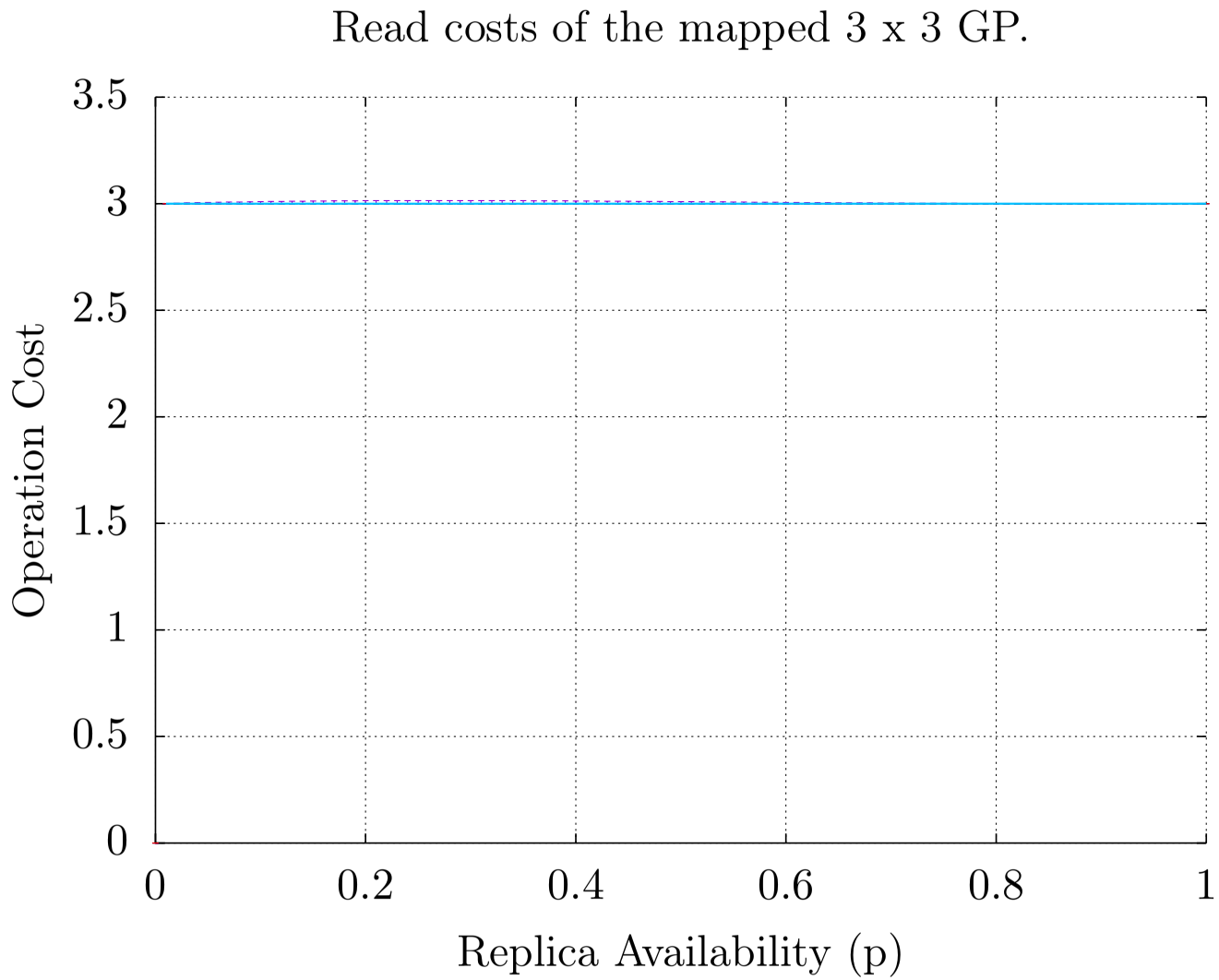


Figure 5.51: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ of the GP with 3×3 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_r(p)$ of the GP with 3×3 replicas in its unmapped state.

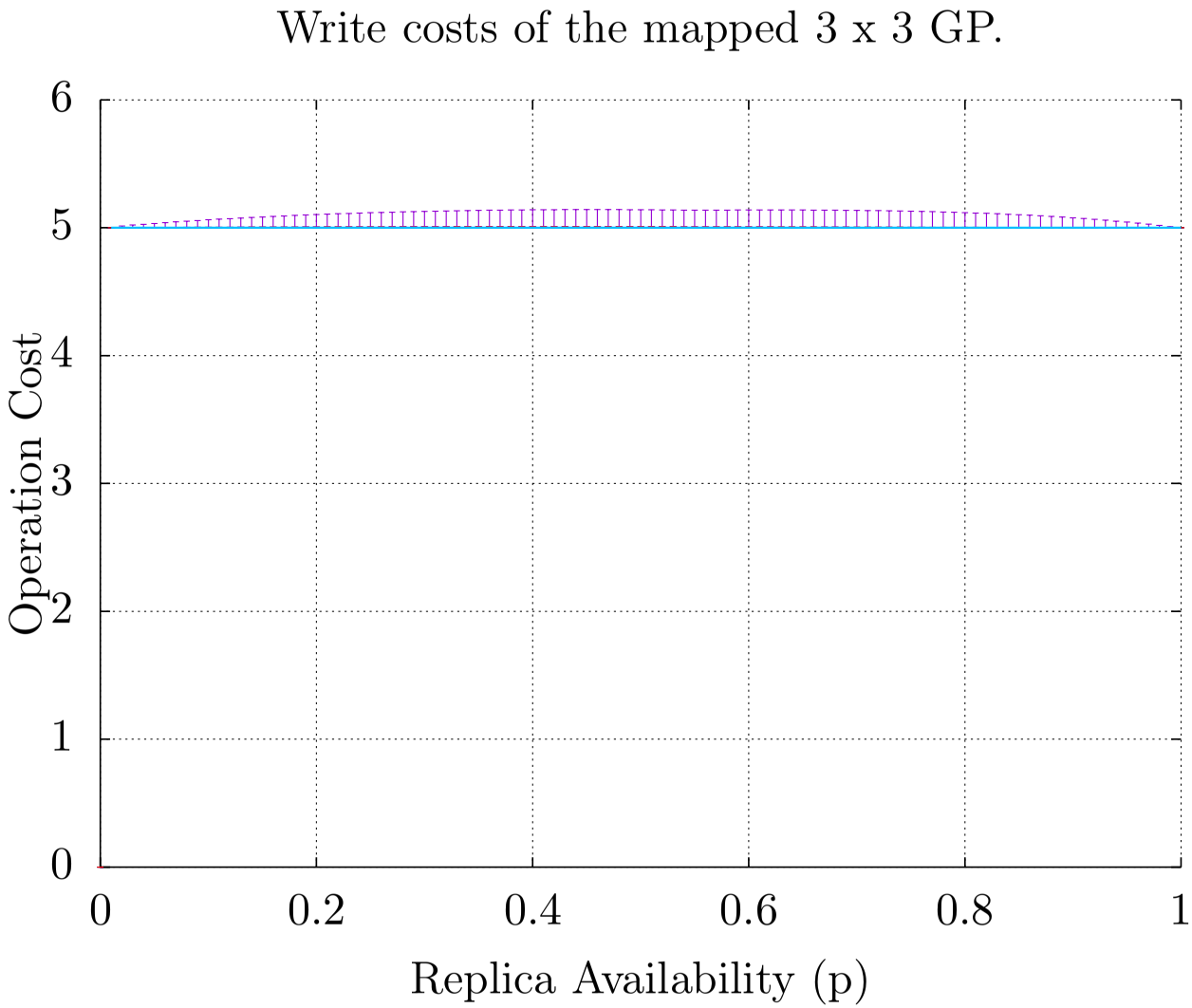


Figure 5.52: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_w(p)$ of the GP with 3×3 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_w(p)$ of the GP with 3×3 replicas in its unmapped state.

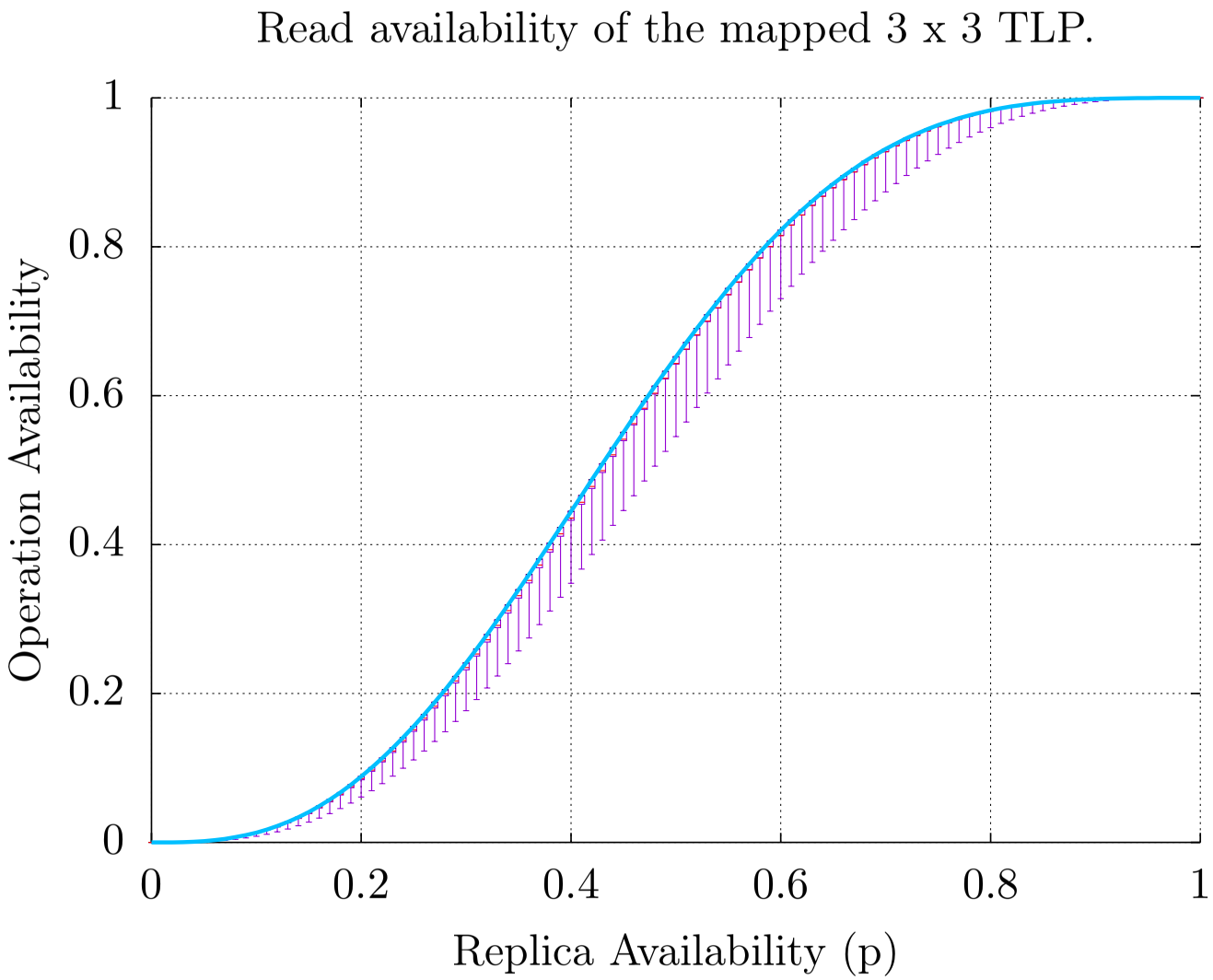


Figure 5.53: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ of the TLP with 3×3 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_r(p)$ of the TLP with 3×3 replicas in its unmapped state.

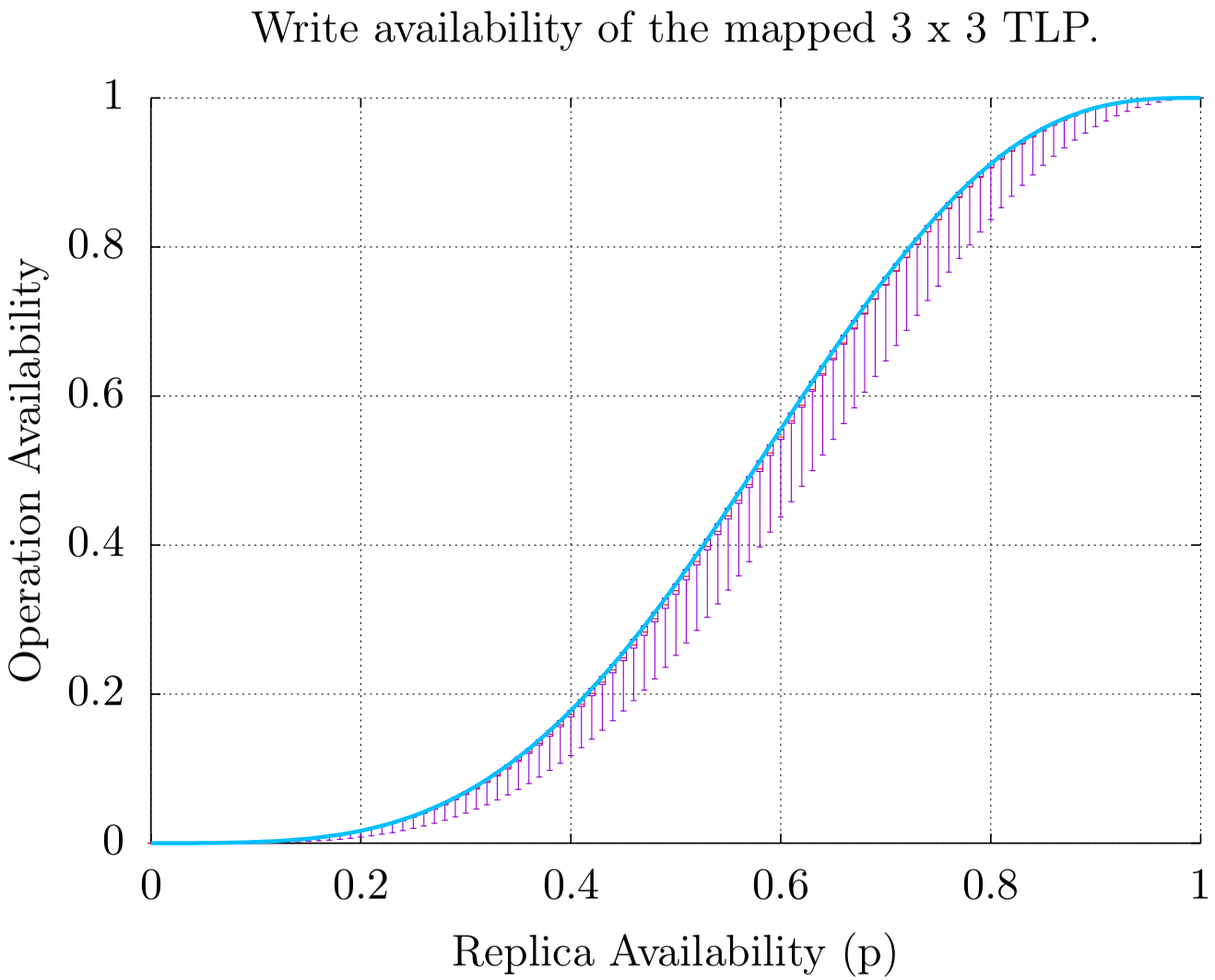


Figure 5.54: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_w(p)$ of the TLP with 3×3 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $a_w(p)$ of the TLP with 3×3 replicas in its unmapped state.

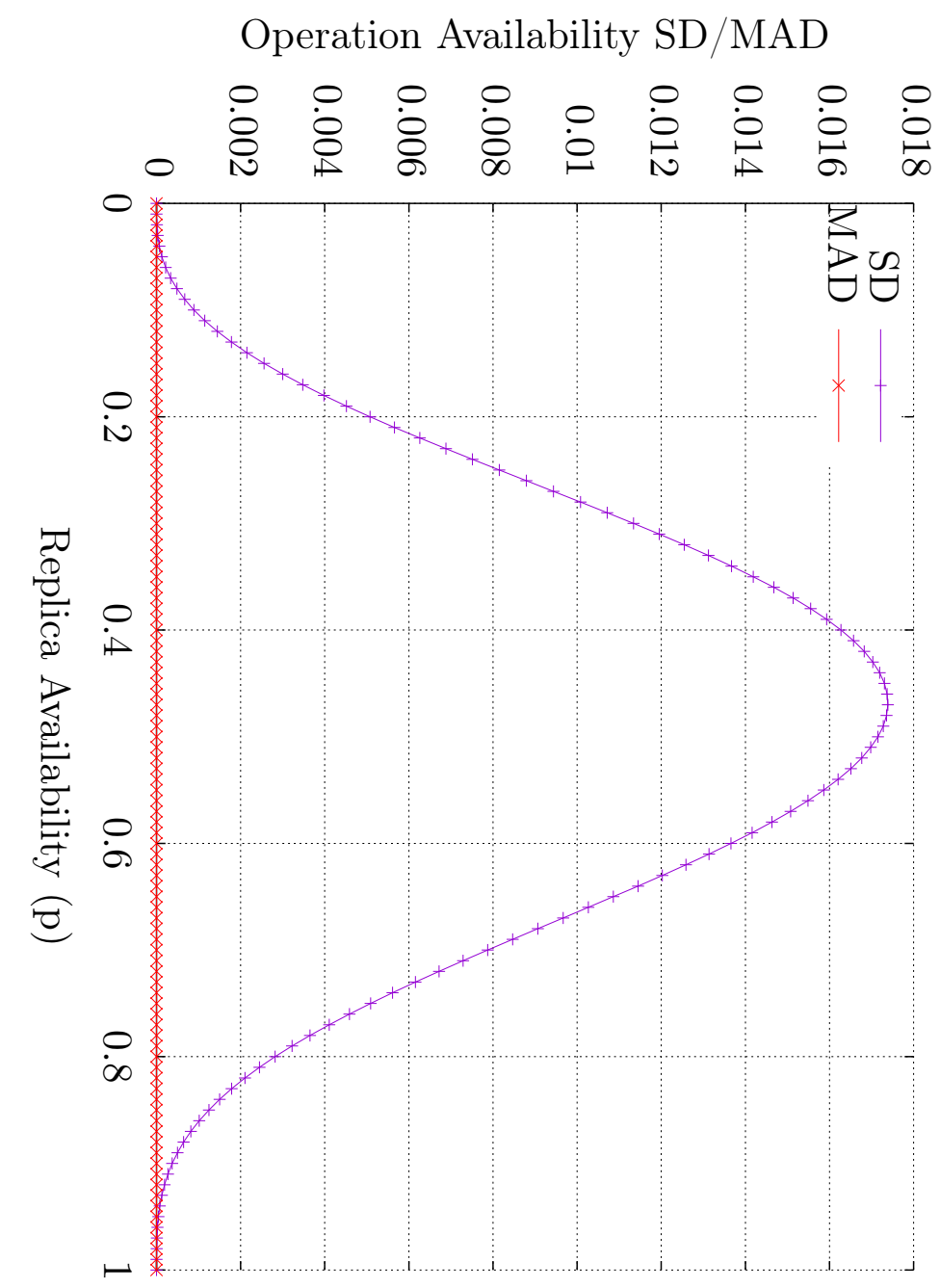


Figure 5.55: The minimal SD and MAD of the $a_r(p)$ of the TLP with 3×3 replicas mapped to 255 simple, non-isomorphic GSs.

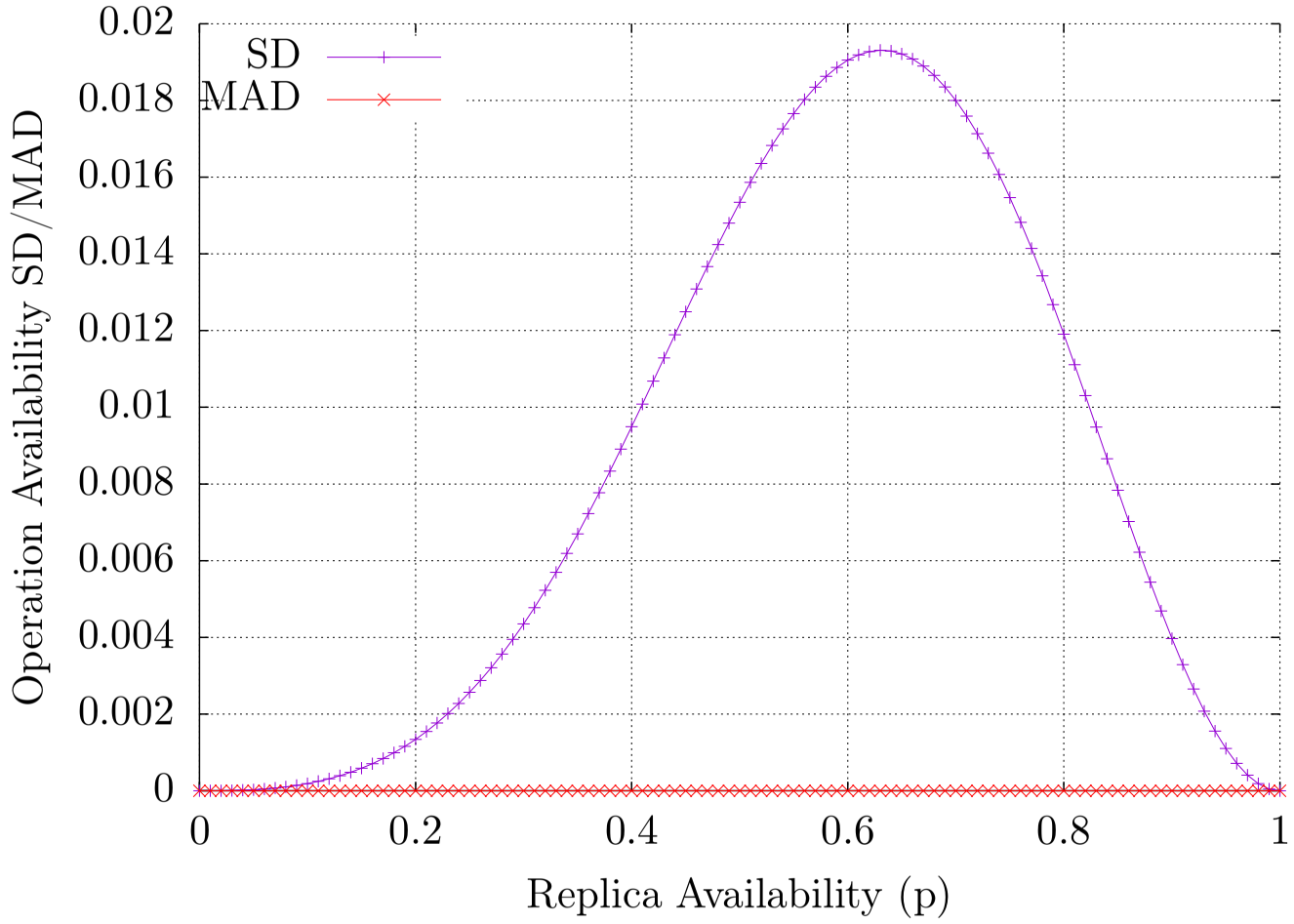


Figure 5.56: The minimal SD and MAD of the $a_w(p)$ of the TLP with 3×3 replicas mapped to 255 simple, non-isomorphic GSs.

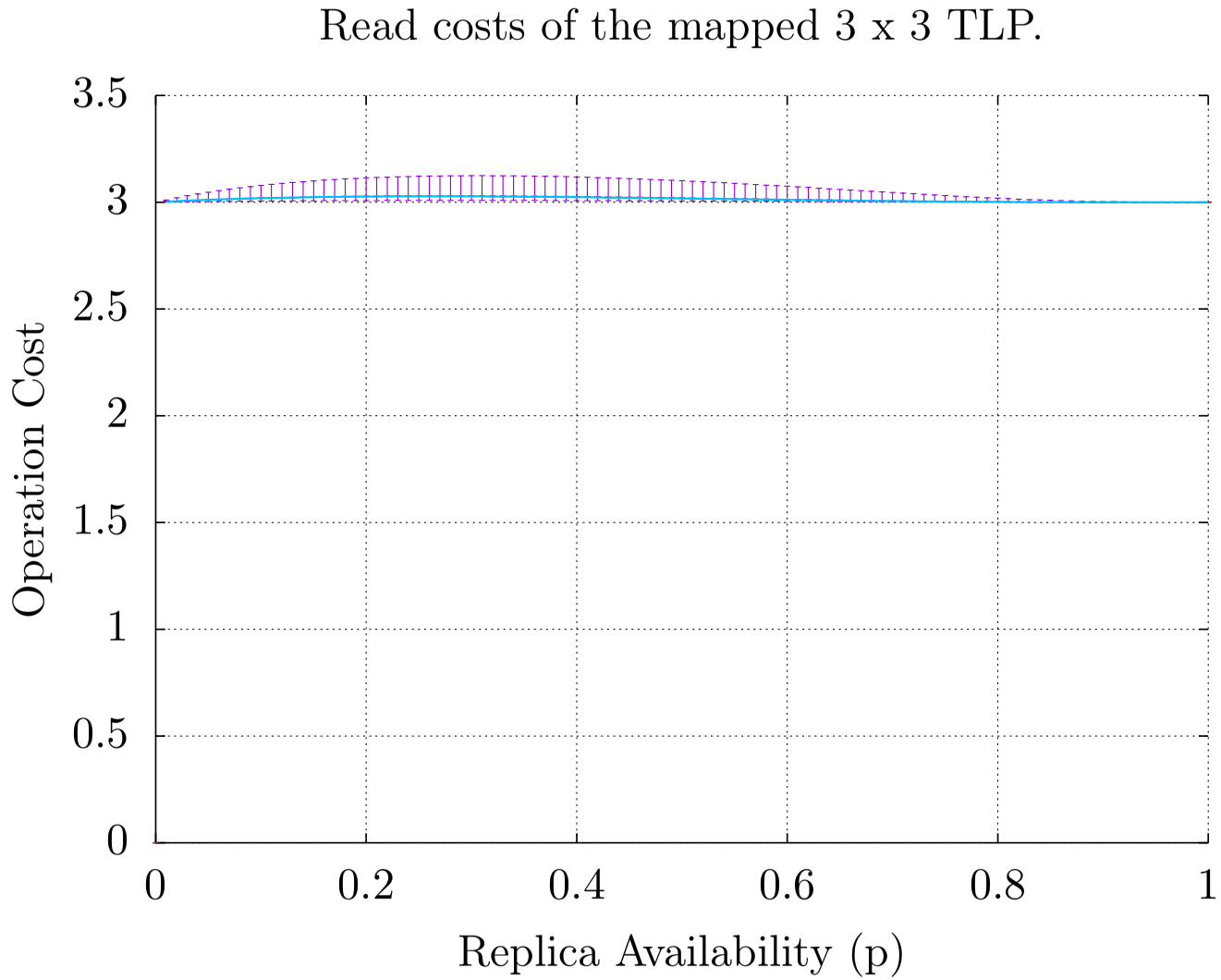


Figure 5.57: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ of the TLP with 3×3 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_r(p)$ of the TLP with 3×3 replicas in its unmapped state.

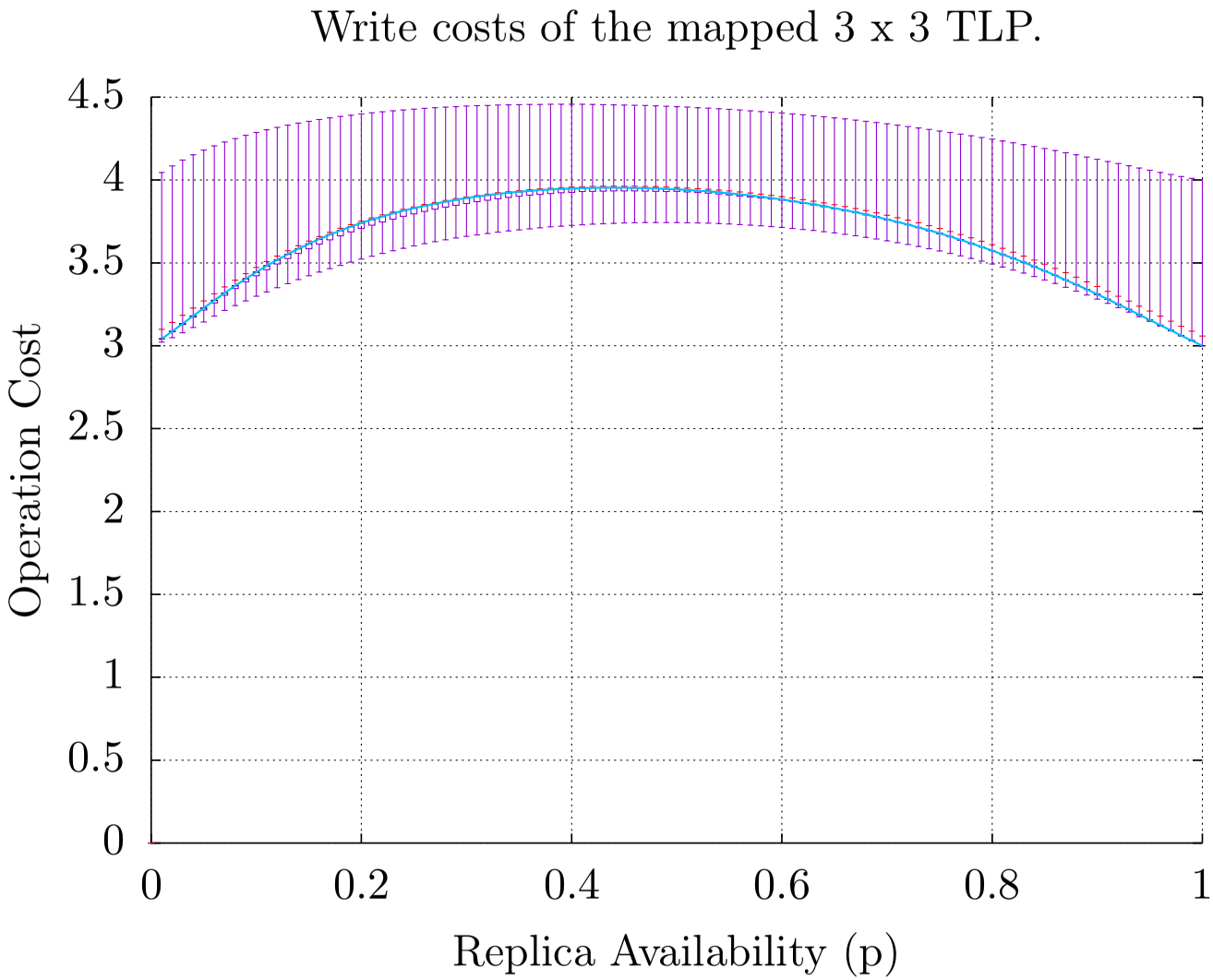


Figure 5.58: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_w(p)$ of the TLP with 3×3 replicas mapped to 255 simple, non-isomorphic GSs. The light blue line is the $c_w(p)$ of the TLP with 3×3 replicas in its unmapped state.

5.5 K-Nearest-Neighbors

The main problem with the evaluation of the mapping approach is that it is computationally expensive: For a given PNT with N vertices $N!$ mappings have to tested. After testing these $N!$ mappings the result can show that the given PNT is not well suited to be for a mapping. This is unfortunate as the computational power to find the best mapping will have already being invested, just to find out that the best mapping is not good enough.

Therefore, the novel approach to use the K-nearest neighbor (kNN) technique to predict the $a_r(p)$, the $a_w(p)$, the $c_r(p)$ and the $c_w(p)$ based on historical results is introduced. This kNN approach is computational less costly in comparison to the mapping approach. If the predicted values are not as good as required, the GS can be discarded as a PNT candidate, thereby saving the computational power required to find the best mapping.

5.5.1 Idea

K-nearest neighbor is a supervised learning classification and regression technique. The term supervised indicates that for the training phase the target value is known. The idea behind the kNN approach is that given existing results, future results can be predicted [19, 20].

The kNN approach is best explained by presenting an example. Let the function $f(x)$ in Figure 5.59 be the system to be predicted. $f(x)$ is unknown. Assuming

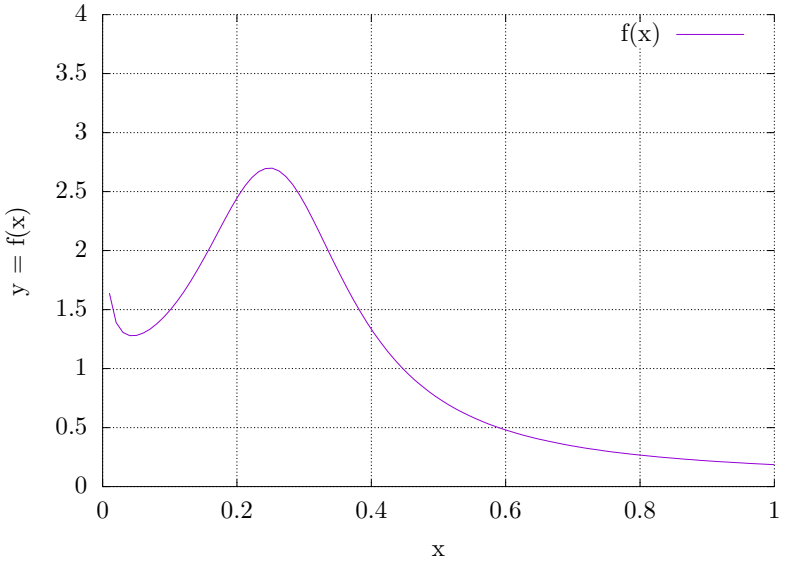


Figure 5.59: The function $f(x)$ shows the behavior of a system. $f(x)$ is unknown.

the historical data for the system is shown in Figure 5.60 on the facing page. The historical data is represented by the red crosses marked on the $f(x)$ function.

Table 5.5 shows the values of the red crosses in Figure 5.60. The goal is now to

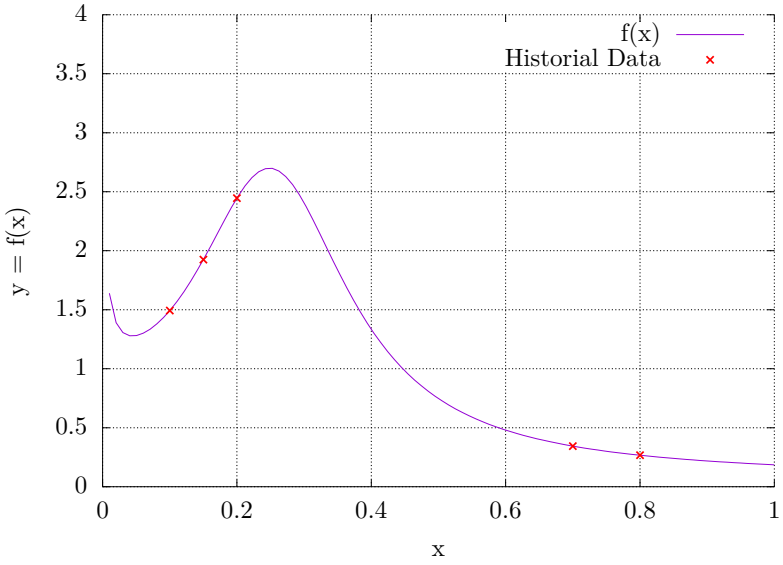


Figure 5.60: Historical data, represented by the red crosses, obtained from the system as shown in Figure 5.59.

x	$y = f(x)$
0.1	1.4926983542521324
0.15	1.9248655564138277
0.2	2.4458290199722015
0.7	0.3442419620570349
0.8	0.2668421585649119

Table 5.5: x and y values of the red crosses in Figure 5.60.

predict the y value for the x value represented by the blue square in Figure 5.61 on the following page[4]. The x value of the blue spuare is 0.24. The idea here is to take the known x values of the historical values and put them in relation to the x of value to predict. Instead of evaluating all x values the k nearest x values are considered. This k is where the k in kNN stems from. The distance measure in this kNN context is the euclidean distance. Therefore, nearest values means, values where the euclidean distance is minimal. As the x values build relations between the data points, x is the feature compared Let $k = 3$ in this example. Table 5.6 on the next page shows the euclidean distances of the x-feature of the historical values in relation to the x-feature of the value that is to be predicted. In this example only one feature is compared. The kNN-approach allows to use an

[4]Even though the blue square is positioned on $y = 0$ the y is not actual 0. The y value is just a stylistic device.

arbitrary amount of features for determining the nearest neighbors. In Table 5.6

x	Euclidean distance to $x = 0.24$
0.1	0.14
0.15	0.09
0.2	0.04
0.7	0.46
0.8	0.56

Table 5.6: Euclidean distances of the x-feature of the historical values.

it can be seen that the x-features $0.1, 0.15$, and 0.2 have the smallest euclidean distance to $x = 0.24$. Therefore, these are the three neighbors that are used to predict the y value of $x = 0.24$. For this example the following is not relevant,

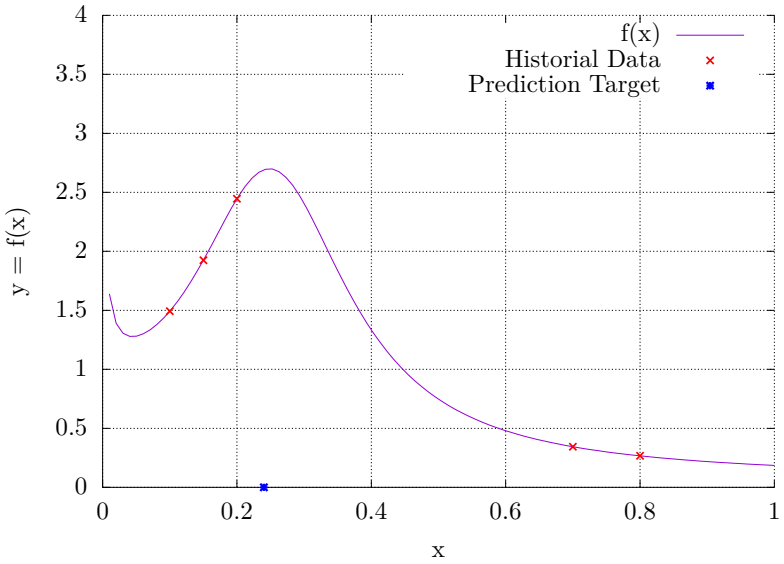


Figure 5.61: The blue square represented the x value of the data that is to be predicted using the kNN approach.

but it is important to gain an understanding of the overall kNN process it is. If more than one feature is used, an additional step is required. This step is called normalization, normalization normalizes the value ranges of the features against each other. Consider the distribution of the two features X and Y in Figure 5.62 on the facing page. The values of the Y-feature are all very close together. The values of the X-feature on the other hand are spread over a much wider value range. When finding the nearest neighbors, the Y-feature plays a much smaller role, as they have, in comparison with the X-feature, much more similar values. To remove this effect the features have to be scaled evenly against each other. This is achieved by dividing all values of a feature by the maximum value of that

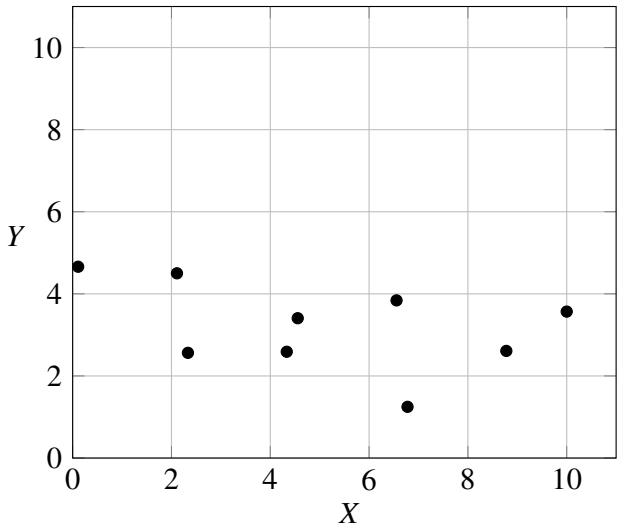


Figure 5.62: A scatterplot of two features X and Y.

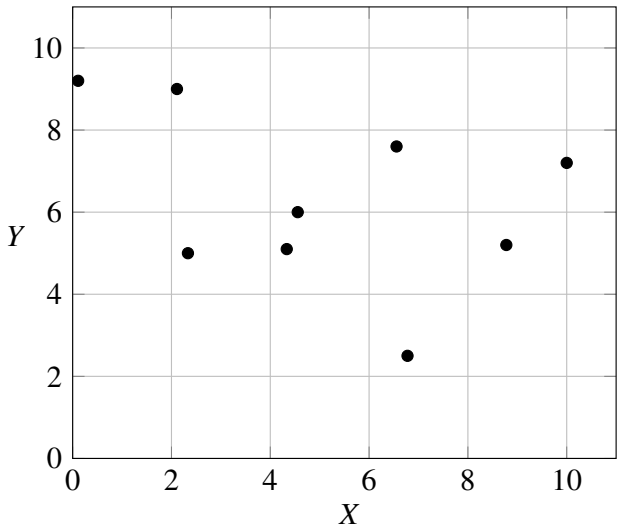


Figure 5.63: A scatterplot of two features X and Y after normalization.

feature[5]. This results in the features being scaled [21, 22]. Figure 5.63 on the previous page shows the result.

As a single prediction value is required the three values have to be aggregated. In this work these aggregation functions are called aggregation functions (AGFs).

In the example, shown in Figure 5.64, the prediction is done with four different AGFs. Eventually, one function has to be chosen to do the prediction, but for analysis purposes looking at different predictions may lead to some more insight into what function is best suited for this purpose. The four AGFs are the min-

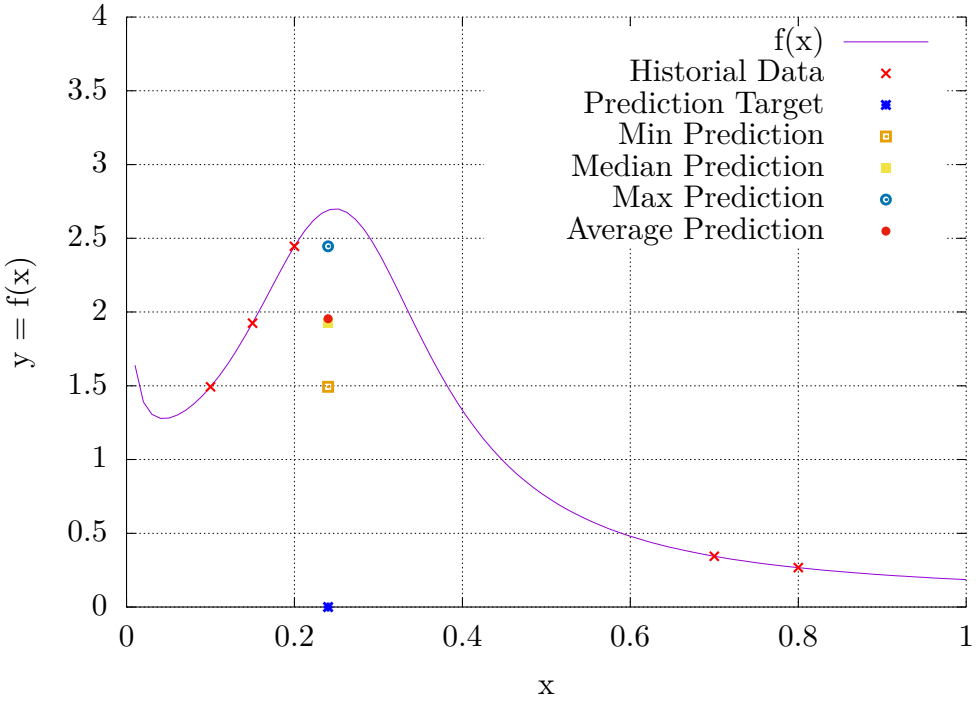


Figure 5.64: Predictions based on the three nearest neighbors. The four AGFs used are the minimum, the maximum, the median, and the average function.

imum function, the maximum function, the median function, and the maximum function. The prediction of the minimum function is shown by the orange box in Figure 5.64. The prediction of the maximum function is shown by the blue circle. The prediction of the median function is shown by the yellow square. Finally, the prediction of the average function is shown by the red dot. The y-value prediction are listed in Table 5.7 on the next page. Figure 5.65 on the facing page shows the predictions as well as the actual y-value. As said before kNN is a supervised method so that in the training phase the value of the prediction target, aka. actual

[5]Here the resulting value of that division is multiplied by 10 to get better readable labels for the axes.

Aggregation function	y-value of prediction
minimum	1.4926983542521324
average	1.9544643102127204
median	1.9248655564138277
maximum	2.4458290199722015

Table 5.7: y-value prediction based on the four used aggregation functions.

y-value is known. The y-value is 2.691686334286658 and is marked by the black

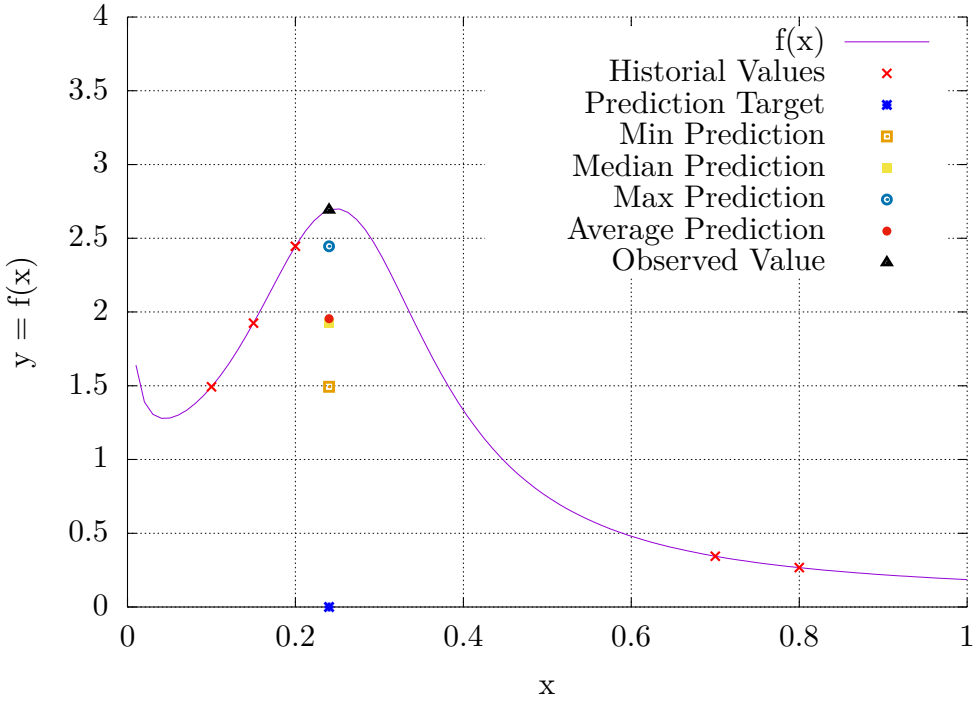


Figure 5.65: Predictions and the actual y-value drawn together. The actual y-value is marked by the black triangle.

triangle. The prediction made of the maximum-function is closed to the black triangle.

The last step is to evaluate the quality of the predictions. The mean squared error (MSE) is a common tool to evaluate predictions. It is calculated as shown below:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{5.29}$$

Where Y is the vector of observed values and $\hat{Y}$ is the vector of predicted values. The MSE has two useful properties. The resulting MSE is always a positive number which simplifies comparison of different MSEs. The other property is that the MSE penalizes larger errors more than smaller errors. Therefore, the smaller the MSE, the better the prediction. Table 5.8 shows the MSEs of the different

Aggregation function	MSE of the prediction
minimum	1.4375721762672724
average	0.5434963127796737
median	0.5880141053774928
maximum	0.0604458190019175

Table 5.8: The MSE of the predictions of the y-value for the $x = 0.24$ value.

aggregation functions used. The maximum-function has the smallest MSE making it the best prediction function in this case. This is in line with the observations from Figure 5.65.

5.5.2 Graph structure (GS) Features

Now, that the prediction approach has been presented, the features used in it have to be defined. As GSs are used as PNTs for the mappings, and their influence on the availability and cost measurements have been shown previously, it is obvious to derive the needed features from the GSs. Not all GS features can be considered as there are just too many. Therefore, the list of investigated GS feature is restricted to those ones that are considered to have merit for the prediction purposes. To

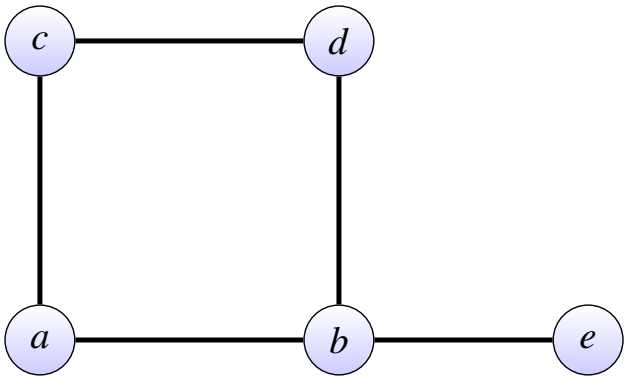


Figure 5.66: The GS used to illustrate the extracted features for the kNN approach.

illustrate the investigated features the GS shown in Figure 5.66 is used. The first feature that was investigated is the order of a GS. The order is equal to $|GS_V|$ where GS is a GS. In short, the order is the number of vertices in a GS. The GS in Figure 5.66 has an order of five [17]. As seen in Section 5.4 on page 55

GSs with different number of vertices will lead to different cost and availability measurements, therefore only use GSs with the same number of vertices are for the predictions. In other words, features of GSs with order N are used to predict the cost and availability measurements of GSs with order N.

The next feature is connectivity. There are two variants of connectivity; vertix connectivity, and edge connectivity. Vertex connectivity defines the minimal number of vertices whose removal partitions the GS [17]. The vertex connectivity of the GS in Figure 5.66 is one, as the removal of vertex b partitions the GS. Edge connectivity defines the minimal number of edges whose removal partition the GS. The GS in Figure 5.66 has an edge connectivity of one. The removal of edge $e_{b,e}$ again partitions the GS. As it is assumed in this work that edges are always available only vertex connectivity are considered. Therefore, the term connectivity is used as a synonym for vertex connectivity.

The following features do not describe the complete GS, but yield individual values for each of the vertices of the GS. As GSs and not vertices of GSs are compared, these vertex features have to be aggregated into GS features. These vertices features aggregated into a GS feature by computing the minimum, the average, the median, the mode, and the maximum of the vertex feature. That means for each of the vertex feature described in the follow five GS features are obtained.

The first vertex feature evaluated is the degree. The degree describes to how many edges a vertex is connected to[17]. Table 5.9 shows the degree of each of the vertices of the GS in Figure 5.66. The information of the Table 5.9 gets

ID	Degree
a	2
b	3
c	2
d	2
e	1

Table 5.9: The degrees of the vertices of the GS shown in Figure 5.66.

transformed into the five GS features shown in Table 5.10.

Feature	value
Minimum Degree	1
Average Degree	2
Median Degree	2
Mode Degree	2
Maximum Degree	3

Table 5.10: The degrees of the vertices of the GS shown in Figure 5.66.

The next vertex feature is the distance between all vertices of the GS. The distance feature described the shortest path between vertices of the GS. The distance counts the number of edges[17]. Let $z = \langle a,b,c \rangle$ be a path then the $\mathbb{E}(z)$ is its distance. Table 5.11 shows the length of the shortest path between each pair of vertices in the GS shown in Figure 5.66. From Table 5.11 five GS features are

ID	a	b	c	d	e
a		1	1	2	2
b			2	1	1
c				1	3
d					2
e					

Table 5.11: Triangle matrix showing the distances between all vertices of the GS in Figure 5.66.

easily computable, as listed in Table 5.12. The distance feature allows to describe

Feature	value
Minimum Distance	1.0
Average Distance	1.7
Median Distance	1.5
Mode Distance	1.0
Maximum Distance	3.0

Table 5.12: The distances of the vertices of the GS shown in Figure 5.66.

how close vertices are to each other in a GS. Especially the distance$_{max}$ measure is interesting. The higher the distance$_{max}$ measurement the more a graph equals a line. A line is easily partitioned, by the removal of a single vertex. Therefore, the distance$_{max}$ measurement is a good indicator how easy a GS can be partitioned. The shorthand distance$_{max}$ symbolizes the Maximum distance. The other shorthands are $_{min}$ for the minimum, $_{avg}$ for the average, $_{md}$ for the mode, $_{mdn}$ for the median, and $_{max}$ for the maximum.

The last vertex feature transformed into a GS feature is the betweenness centrality (BC) measure [23]. The BC feature makes a statement about how often a particular vertex is part of all shortest paths through a graph. The BC of a vertex v is defined by the function $g(v)$ as shown below.

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{5.30}$$

Where $\sigma_{st}(v)$ donates the number of shortest paths between the vertices s and t where the vertex v is a part of. The divisor σ_{st} represents the number of shortest paths between the vertices s and t. σ_{st} is a list of all shortest paths between all

pairs of vertices. There can be more than one shortest path, for example the path (c,a,b) has the same length as the path (c,d,b). Therefore, both are part of the set of shortest paths between the vertices c and b [24]. Table 5.13 lists all shortest paths between all the vertices in GS in Figure 5.66. Table 5.14 shows the number

from	to	shortest paths
a	b	$\langle a,b\rangle$
a	c	$\langle a,c\rangle$
a	d	$\langle a,c,d\rangle$
		$\langle a,b,d\rangle$
a	e	$\langle a,b,e\rangle$
b	c	$\langle b,a,c\rangle$
		$\langle b,d,c\rangle$
b	d	$\langle b,d\rangle$
b	e	$\langle b,e\rangle$
c	d	$\langle c,d\rangle$
c	e	$\langle c,a,b,e\rangle$
		$\langle c,d,b,e\rangle$
d	e	$\langle d,b,e\rangle$

Table 5.13: All shortest path between all pairs of vertices of the GS shown in Figure 5.66.

of occurrences of each of the vertex in GS in all shortest path, as well as their BC value based on the Table 5.13. Table 5.15 on the following page shows the five GS

s	t	σ_{st}	$\sigma_{st}(a)$	$\sigma_{st}(b)$	$\sigma_{st}(c)$	$\sigma_{st}(d)$	$\sigma_{st}(e)$
a	b	1					
a	c	1					
a	d	2		1	1		
a	e	1		1			
b	c	2	1			1	
b	d	1					
b	e	1					
c	d	1					
c	e	2	1	2		1	
d	e	1		1			
$\sum \sigma_{st}$ / $\sigma_{st}(x)$			1.0	3.5	0.5	1.0	0.0

Table 5.14: The number of occurrences of each vertex of the GS in Figure 5.66 in all shortest paths. The number 0 is omitted for better readability, except in the result row.

features extracted from the vertex feature shown in Table 5.14.

The following Table 5.16 shows all the different features evaluated. Testing

Feature	value
Minimum BC	0.0
Average BC	1.2
Median BC	1.0
Mode BC	1.0
Maximum BC	3.5

Table 5.15: The BCs of the vertices of the GS shown in Figure 5.66.

Feature
Minimum Degree
Average Degree
Median Degree
Mode Degree
Maximum Degree
Minimum Distance
Average Distance
Median Distance
Mode Distance
Maximum Distance
Minimum BC
Average BC
Median BC
Mode BC
Maximum BC
Connectivity

Table 5.16: All evaluated features.

each feature individual might not result in the best possible prediction, therefore also sets of different features are tested together. The question arises, which combinations of features should be tested, as testing all combinations is practically impossible. It is impossible, as there are 16 individual features resulting in a $16! = 20922789888000$ combinations to test. Two reduce the number of combinations features are grouped into classes. Classes are based on the underlying GS feature that they represent. This leaves the following four classes of features:

- Degree class
- Distance class
- BC class
- Connectivity class.

Now only one feature of each group is allowed to appear in any set of features used for the prediction. This leaves $5 \cdot 5 \cdot 5 \cdot 1 = 125$ combinations of features to test.

5.5.3 Cross Validation

One of the problems with machine learning approaches is what is known as overfitting. Overfitting basically describes a state where the given trainings data is so similar to testing data that the resulting prediction model is unable to predicate data items that has different characteristic in comparison to the trainings data [25].

A common approach to mitigate this problem is to use a technique called cross-validation (CV) [26]. The idea of CV is to partition the available data into a number of N equal sized distinct sets and then use $N-1$ sets as training data, and use the remaining set to test the prediction quality. This is repeated N times and the prediction results are aggregated. Figure 5.67 shows an example, where the

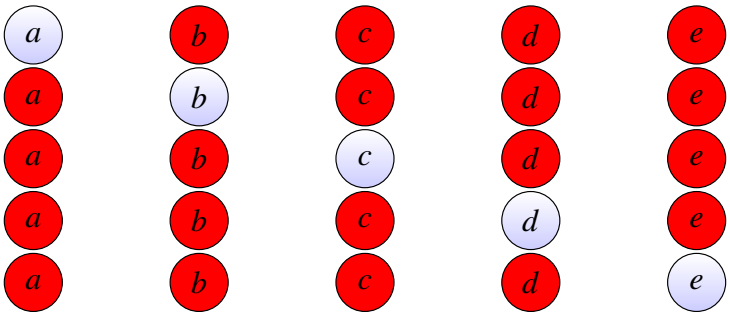


Figure 5.67: Training and test data sequence of the CV approach. Each line represents one test run. Blue circles represent test data sets, red circles represent training data sets.

available data is separated into five sets. In each row the four red circles represent

the data sets that are used for the training of the kNN approach and the blue circle represents the test data that evaluates the prediction based on the training data. The individual MSEs of each the tests get summarized, producing the final MSE.

5.5.4 Evaluation

In the evaluation of the predictions done with the kNN approach different factors are considered. The kNN approach is tested with $k \in 2,3,5,7$. As input to the CV the mapping resulting from the evaluation of Section 5.4 are used. Table 5.17 ... Table 5.20 show the MSE of the predictions of the kNN approach of the mapping of the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, the $c_w(p)$ based on two neighbors.

The header of each table shows the AGFs used. Below each AGF, a column shows the MSEs and the other column shows an ID. Each row shows the results for one GP for all the AGFs. The table Subsection 5.5.4 is used as an example to explain interpretation of the results. In this table, only highlighted parts are

AGFs / QPs	Min		Average		Median		Mode		Max	
	MSE	ID	MSE	ID	MSE	ID	MSE	ID	MSE	ID
GP 4×2	7.62	(1)	0.00	(2)	0.00	(2)	0.77	(3)	8.81	(4)
GP 2×4	46.55	(5)	0.00	(2)	0.00	(2)	3.40	(6)	61.46	(7)
MCS	130.92	(1)	0.00	(2)	0.00	(2)	3.66	(3)	149.22	(8)
TLP 2×4	0.68	(9)	0.00	(2)	0.00	(2)	0.68	(9)	0.45	(10)
TLP 4×2	0.23	(11)	0.00	(2)	0.00	(2)	0.23	(11)	0.23	(12)

considered. The value 3.40 is the MSE of the prediction by the kNN approach for the QP on a 2×4 grid LNT. The value 6 is the ID of the feature set used in this prediction. The AGF used is the mode function. The number of neighbors, and the number of replicas is listed in each caption of each result table.

As the four tables show, the MSE for the AGFs average, and median is 0.00. That means, the predictions are 100% correct. Table 5.22 on page 124 shows the sets of features that lead to at least one best prediction. The ID field in this table corresponds to the ID field of the four Tables 5.17, 5.18, 5.19, and 5.20. The column Occurrences lists how often a set of features made the best prediction. The feature BC_{Max} was able to perfectly predict the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ of all five tested QPs. BC is also part of most of the other feature sets shown in Table 5.22. The results of the predictions for QPs with eight replicas with three, and five neighbors are omitted as they show the same MSEs and use the same feature sets. The same is true for the test with seven neighbors. For completeness, the $a_r(p)$, and the $a_w(p)$ of that test are shown in Table 5.23 on page 125 and Table 5.24 on page 125. The feature sets leading to the best predictions with seven neighbors is shown in Table 5.26 on page 126. The shown results are equal to Table 5.22. QPs with nine replicas were also tested. Table 5.27 on page 127 to Table 5.30 on page 127 show the MSE of these predictions. Again, first the predictions based on two neighbors are shown.

AGFs / QPs	Min		Average		Median		Mode		Max	
	MSE	ID	MSE	ID	MSE	ID	MSE	ID	MSE	ID
GP 4×2	7.62	(1)	0.00	(2)	0.00	(2)	0.77	(3)	8.81	(4)
GP 2×4	46.55	(5)	0.00	(2)	0.00	(2)	3.40	(6)	61.46	(7)
MCS	130.92	(1)	0.00	(2)	0.00	(2)	3.66	(3)	149.22	(8)
TLP 2×4	0.68	(9)	0.00	(2)	0.00	(2)	0.68	(9)	0.45	(10)
TLP 4×2	0.23	(11)	0.00	(2)	0.00	(2)	0.23	(11)	0.23	(12)

Table 5.17: The MSE of the $a_r(p)$ predictions by the kNN approach with eight replicas and $k = 2$.

AGFs / QPs	Min		Average		Median		Mode		Max	
	MSE	ID	MSE	ID	MSE	ID	MSE	ID	MSE	ID
GP 4×2	9.09	(13)	0.00	(2)	0.00	(2)	0.89	(14)	17.12	(12)
GP 2×4	52.59	(15)	0.00	(2)	0.00	(2)	1.09	(16)	114.04	(1)
MCS	41.12	(17)	0.00	(2)	0.00	(2)	2.00	(18)	69.21	(12)
TLP 2×4	69.97	(16)	0.00	(2)	0.00	(2)	1.05	(19)	209.22	(20)
TLP 4×2	42.93	(21)	0.00	(2)	0.00	(2)	0.67	(22)	103.20	(4)

Table 5.18: The MSE of the $a_w(p)$ predictions by the kNN approach with eight replicas and $k = 2$.

AGFs / QPs	Min		Average		Median		Mode		Max	
	MSE	ID	MSE	ID	MSE	ID	MSE	ID	MSE	ID
GP 4×2	0.00	(23)	0.00	(24)	0.00	(24)	0.00	(23)	0.00	(25)
GP 2×4	0.00	(24)	0.00	(24)	0.00	(24)	0.00	(24)	0.00	(24)
MCS	0.00	(24)	0.00	(24)	0.00	(24)	0.00	(24)	0.00	(24)
TLP 2×4	0.00	(18)	0.00	(2)	0.00	(2)	0.00	(18)	0.00	(10)
TLP 4×2	0.00	(11)	0.00	(24)	0.00	(24)	0.00	(11)	0.00	(26)

Table 5.19: The MSE of the $c_r(p)$ predictions by the kNN approach with eight replicas and $k = 2$.

AGFs / QPs	Min		Average		Median		Mode		Max	
	MSE	ID	MSE	ID	MSE	ID	MSE	ID	MSE	ID
GP 4×2	0.00	(19)	0.00	(2)	0.00	(2)	0.00	(19)	0.00	(19)
GP 2×4	0.99	(27)	0.00	(2)	0.00	(2)	0.01	(28)	0.59	(29)
MCS	0.00	(4)	0.00	(2)	0.00	(2)	0.00	(4)	0.00	(30)
TLP 2×4	0.79	(31)	0.00	(2)	0.00	(2)	0.02	(32)	0.31	(33)
TLP 4×2	1.56	(34)	0.00	(2)	0.00	(2)	0.01	(14)	0.39	(18)

Table 5.20: The MSE of the $c_w(p)$ predictions by the kNN approach with eight replicas and $k = 2$.

ID	Set of features	Occurrences
(1)	{ BetweenneesAverage, DegreeMix }	3
(2)	{ BetweenneesMax }	32
(3)	{ BetweenneesMax, Connectivity, DegreeMax }	2
(4)	{ BetweenneesAverage, DiameterMedian, DegreeMix }	4
(5)	{ BetweenneesMode, DiameterAverage, Connectivity, DegreeMedian }	1
(6)	{ BetweenneesMedian, DiameterAverage, DegreeMax }	1
(7)	{ BetweenneesMax, DiameterAverage, Connectivity, DegreeAverage }	1
(8)	{ BetweenneesMix, DiameterAverage, DegreeMedian }	1
(9)	{ BetweenneesAverage, DiameterAverage, Connectivity, DegreeAverage }	2
(10)	{ BetweenneesMax, DiameterAverage }	2
(11)	{ BetweenneesAverage, Connectivity }	4
(12)	{ BetweenneesMax, DiameterMax, Connectivity, DegreeMode }	3
(13)	{ DiameterAverage, DegreeMix }	1
(14)	{ BetweenneesMedian, Connectivity, DegreeAverage }	2
(15)	{ BetweenneesAverage, DiameterMax, Connectivity, DegreeMix }	1
(16)	{ BetweenneesMode, DiameterAverage, Connectivity, DegreeMode }	2
(17)	{ BetweenneesMode, DiameterAverage, Connectivity, DegreeAverage }	1
(18)	{ BetweenneesMedian, DiameterAverage, DegreeMix }	4
(19)	{ DiameterAverage, Connectivity, DegreeAverage }	4
(20)	{ BetweenneesMedian, Connectivity }	1
(21)	{ BetweenneesMedian, DiameterAverage, DegreeAverage }	1
(22)	{ BetweenneesMedian, DegreeAverage }	1
(23)	{ DegreeAverage }	2
(24)	{ BetweenneesMedian }	14
(25)	{ BetweenneesMedian, DiameterAverage }	1
(26)	{ BetweenneesMax, DiameterMax, DegreeMode }	1
(27)	{ BetweenneesMix, DiameterAverage, Connectivity, DegreeMedian }	1
(28)	{ BetweenneesMedian, DiameterMax, DegreeMode }	1
(29)	{ BetweenneesMix, DiameterMax, DegreeAverage }	1
(30)	{ BetweenneesAverage, DiameterMedian, Connectivity, DegreeMix }	1
(31)	{ BetweenneesMax, DiameterMax, DegreeMedian }	1
(32)	{ BetweenneesMax, DiameterMax, Connectivity, DegreeMedian }	1
(33)	{ BetweenneesMedian, DiameterAverage, Connectivity }	1
(34)	{ BetweenneesAverage, DiameterMax, Connectivity, DegreeMode }	1

Table 5.22: The graph properties and graph property combinations used in the kNN where $k = 2$ predictions that lead to the best predictions in at least one instance with eight replicas.

AGFs / QPs	Min		Average		Median		Mode		Max	
	MSE	ID	MSE	ID	MSE	ID	MSE	ID	MSE	ID
GP 4×2	7.62	(1)	0.00	(2)	0.00	(2)	0.77	(3)	8.81	(4)
GP 2×4	46.55	(5)	0.00	(2)	0.00	(2)	3.40	(6)	61.46	(7)
MCS	130.92	(1)	0.00	(2)	0.00	(2)	3.66	(3)	149.22	(8)
TLP 2×4	0.68	(9)	0.00	(2)	0.00	(2)	0.68	(9)	0.45	(10)
TLP 4×2	0.23	(11)	0.00	(2)	0.00	(2)	0.23	(11)	0.23	(12)

Table 5.23: The MSE of the $a_r(p)$ predictions by the kNN approach with eight replicas and $k = 7$.

AGFs / QPs	Min		Average		Median		Mode		Max	
	MSE	ID	MSE	ID	MSE	ID	MSE	ID	MSE	ID
GP 4×2	9.09	(13)	0.00	(2)	0.00	(2)	0.89	(14)	17.12	(12)
GP 2×4	52.59	(15)	0.00	(2)	0.00	(2)	1.09	(16)	114.04	(1)
MCS	41.12	(17)	0.00	(2)	0.00	(2)	2.00	(18)	69.21	(12)
TLP 2×4	69.97	(16)	0.00	(2)	0.00	(2)	1.05	(19)	209.22	(20)
TLP 4×2	42.93	(21)	0.00	(2)	0.00	(2)	0.67	(22)	103.20	(4)

Table 5.24: The MSE of the $a_w(p)$ predictions by the kNN approach with eight replicas and $k = 7$.

Interestingly, the predictions are already perfect. The AGFs average and median have a MSE of 0.00. Table 5.32 on page 128 shows the feature sets that lead to the best predictions. BC_{Max} is again dominant. Again the results of the tests with three, and five neighbors are omitted as they are equal to the test results with two neighbors. To emphasize this fact, the results of the test with seven neighbors are shown. The results are shown in Table 5.33 on page 128 and Table 5.34 on page 129.

BC, especially BC_{Max} is the dominant feature in the analysis. The dominance of the BetweennessMax can be explained by looking into its meaning. If quorums are no longer connected after they have been mapped to a PNT, the mapping approach will reconnect the replicas of the quorum with as little additional replicas as possible. These replicas are found by finding the shortest paths between not connect vertices in the PNT. The BC property makes a statement about how often a particular replica is part of all shortest paths through a graph. BetweennessMax expresses how often the replica/vertex that is part of the most shortest paths in a graph is part of a shortest path. Therefore, BetweennessMax basically states the BC value of the most important replica in the graph in regards to quorums mappings. As graphs with the same number of replicas for each kNN iteration are used, BetweennessMax turns out to be a very good estimator for the quality of the mapping, that can be expected from a graph. This is because graphs with the same BetweennessMax value have a very similar structure.

For example, consider all shortest path of the GS in Figure 5.66 as shown in Table 5.13 leading to the BC values as shown in Table 5.14. Replica b is part of nine of the 13 shortest paths, leading to a BC value of 3.5. It is also the highest

ID	Set of features	Occurrences
(1)	{ BetweenneesAverage, DegreeMix }	3
(2)	{ BetweenneesMax }	32
(3)	{ BetweenneesMax, Connectivity, DegreeMax }	2
(4)	{ BetweenneesAverage, DiameterMedian, DegreeMix }	4
(5)	{ BetweenneesMode, DiameterAverage, Connectivity, DegreeMedian }	1
(6)	{ BetweenneesMedian, DiameterAverage, DegreeMax }	1
(7)	{ BetweenneesMax, DiameterAverage, Connectivity, DegreeAverage }	1
(8)	{ BetweenneesMix, DiameterAverage, DegreeMedian }	1
(9)	{ BetweenneesAverage, DiameterAverage, Connectivity, DegreeAverage }	2
(10)	{ BetweenneesMax, DiameterAverage }	2
(11)	{ BetweenneesAverage, Connectivity }	4
(12)	{ BetweenneesMax, DiameterMax, Connectivity, DegreeMode }	3
(13)	{ DiameterAverage, DegreeMix }	1
(14)	{ BetweenneesMedian, Connectivity, DegreeAverage }	2
(15)	{ BetweenneesAverage, DiameterMax, Connectivity, DegreeMix }	1
(16)	{ BetweenneesMode, DiameterAverage, Connectivity, DegreeMode }	2
(17)	{ BetweenneesMode, DiameterAverage, Connectivity, DegreeAverage }	1
(18)	{ BetweenneesMedian, DiameterAverage, DegreeMix }	4
(19)	{ DiameterAverage, Connectivity, DegreeAverage }	4
(20)	{ BetweenneesMedian, Connectivity }	1
(21)	{ BetweenneesMedian, DiameterAverage, DegreeAverage }	1
(22)	{ BetweenneesMedian, DegreeAverage }	1
(23)	{ DegreeAverage }	2
(24)	{ BetweenneesMedian }	14
(25)	{ BetweenneesMedian, DiameterAverage }	1
(26)	{ BetweenneesMax, DiameterMax, DegreeMode }	1
(27)	{ BetweenneesMix, DiameterAverage, Connectivity, DegreeMedian }	1
(28)	{ BetweenneesMedian, DiameterMax, DegreeMode }	1
(29)	{ BetweenneesMix, DiameterMax, DegreeAverage }	1
(30)	{ BetweenneesAverage, DiameterMedian, Connectivity, DegreeMix }	1
(31)	{ BetweenneesMax, DiameterMax, DegreeMedian }	1
(32)	{ BetweenneesMax, DiameterMax, Connectivity, DegreeMedian }	1
(33)	{ BetweenneesMedian, DiameterAverage, Connectivity }	1
(34)	{ BetweenneesAverage, DiameterMax, Connectivity, DegreeMode }	1

Table 5.26: The graph properties and graph property combinations used in the kNN where $k = 7$ predictions that lead to the best predictions in at least one instance with eight replicas.

AGFs / QPs	Min		Average		Median		Mode		Max	
	MSE	ID	MSE	ID	MSE	ID	MSE	ID	MSE	ID
GP 3×3	27.29	(1)	0.00	(2)	0.00	(2)	2.70	(3)	62.76	(4)
MCS	84.17	(1)	0.00	(2)	0.00	(2)	2.99	(5)	175.70	(4)
TLP 3×3	8.34	(6)	0.00	(2)	0.00	(2)	1.07	(7)	66.49	(8)

Table 5.27: The MSE of the $a_r(p)$ predictions by the kNN approach with nine replicas and $k=2$.

AGFs / QPs	Min		Average		Median		Mode		Max	
	MSE	ID	MSE	ID	MSE	ID	MSE	ID	MSE	ID
GP 3×3	47.34	(1)	0.00	(2)	0.00	(2)	2.25	(9)	276.35	(10)
MCS	84.17	(1)	0.00	(2)	0.00	(2)	2.99	(5)	175.70	(4)
TLP 3×3	8.66	(11)	0.00	(2)	0.00	(2)	0.86	(7)	138.32	(12)

Table 5.28: The MSE of the $a_w(p)$ predictions by the kNN approach with nine replicas and $k=2$.

AGFs / QPs	Min		Average		Median		Mode		Max	
	MSE	ID	MSE	ID	MSE	ID	MSE	ID	MSE	ID
GP 3×3	0.00	(13)	0.00	(2)	0.00	(2)	0.00	(13)	0.00	(14)
MCS	0.00	(4)	0.00	(2)	0.00	(2)	0.00	(4)	0.00	(15)
TLP 3×3	0.01	(16)	0.00	(2)	0.00	(2)	0.00	(17)	0.01	(18)

Table 5.29: The MSE of the $c_r(p)$ predictions by the kNN approach with nine replicas and $k=2$.

AGFs / QPs	Min		Average		Median		Mode		Max	
	MSE	ID	MSE	ID	MSE	ID	MSE	ID	MSE	ID
GP 3×3	0.02	(8)	0.00	(2)	0.00	(2)	0.01	(19)	0.01	(20)
MCS	0.00	(4)	0.00	(2)	0.00	(2)	0.00	(4)	0.00	(15)
TLP 3×3	3.55	(21)	0.00	(2)	0.00	(2)	0.00	(22)	1.71	(23)

Table 5.30: The MSE of the $c_w(p)$ predictions by the kNN approach with nine replicas and $k=2$.

ID	Set of features	Occurrences
(1)	{ BetweenneesAverage, DiameterMax, Connectivity, DegreeMedian }	4
(2)	{ BetweenneesMax }	24
(3)	{ BetweenneesAverage, DiameterAverage, Connectivity, DegreeAverage }	1
(4)	{ BetweenneesMode, DiameterAverage, DegreeAverage }	7
(5)	{ BetweenneesMedian, DiameterMax, Connectivity, DegreeMax }	2
(6)	{ BetweenneesMode, DiameterAverage, DegreeMedian }	1
(7)	{ BetweenneesMedian, DiameterAverage, Connectivity }	2
(8)	{ BetweenneesMedian, DiameterMax, DegreeAverage }	2
(9)	{ BetweenneesMax, DiameterAverage, Connectivity, DegreeMode }	1
(10)	{ BetweenneesMix, DiameterAverage, DegreeMedian }	1
(11)	{ BetweenneesMedian, DiameterAverage, Connectivity, DegreeMedian }	1
(12)	{ BetweenneesMix, DiameterMax, DegreeAverage }	1
(13)	{ BetweenneesAverage, DiameterMax, Connectivity, DegreeMax }	2
(14)	{ BetweenneesMax, Connectivity, DegreeMedian }	1
(15)	{ BetweenneesMode, DiameterAverage, Connectivity, DegreeMix }	2
(16)	{ BetweenneesMedian, DegreeMedian }	1
(17)	{ BetweenneesAverage, DiameterMax, DegreeMode }	1
(18)	{ BetweenneesMode, DegreeAverage }	1
(19)	{ BetweenneesMedian, DiameterAverage, Connectivity, DegreeMode }	1
(20)	{ BetweenneesAverage, Connectivity, DegreeMedian }	1
(21)	{ BetweenneesMedian, DegreeAverage }	1
(22)	{ BetweenneesMax, DiameterMax, Connectivity, DegreeAverage }	1
(23)	{ BetweenneesMedian, DiameterAverage, DegreeMode }	1

Table 5.32: The graph properties and graph property combinations used in the kNN where $k = 2$ predictions that lead to the best predictions in at least one instance with nine replicas.

AGFs / QPs	Min		Average		Median		Mode		Max	
	MSE	ID	MSE	ID	MSE	ID	MSE	ID	MSE	ID
GP 3×3	27.29	(1)	0.00	(2)	0.00	(2)	2.70	(3)	62.76	(4)
MCS	84.17	(1)	0.00	(2)	0.00	(2)	2.99	(5)	175.70	(4)
TLP 3×3	8.34	(6)	0.00	(2)	0.00	(2)	1.07	(7)	66.49	(8)

Table 5.33: The MSE of the $a_r(p)$ predictions by the kNN approach with nine replicas and $k = 7$.

QPs \ AGFs	Min		Average		Median		Mode		Max	
	MSE	ID	MSE	ID	MSE	ID	MSE	ID	MSE	ID
GP 3×3	47.34	(1)	0.00	(2)	0.00	(2)	2.25	(9)	276.35	(10)
MCS	84.17	(1)	0.00	(2)	0.00	(2)	2.99	(5)	175.70	(4)
TLP 3×3	8.66	(11)	0.00	(2)	0.00	(2)	0.86	(7)	138.32	(12)

Table 5.34: The MSE of the $a_w(p)$ predictions by the kNN approach with nine replicas and $k = 7$.

ID	Set of features	Occurrences
(1)	{ BetweenneesAverage, DiameterMax, Connectivity, DegreeMedian }	4
(2)	{ BetweenneesMax }	24
(3)	{ BetweenneesAverage, DiameterAverage, Connectivity, DegreeAverage }	1
(4)	{ BetweenneesMode, DiameterAverage, DegreeAverage }	7
(5)	{ BetweenneesMedian, DiameterMax, Connectivity, DegreeMax }	2
(6)	{ BetweenneesMode, DiameterAverage, DegreeMedian }	1
(7)	{ BetweenneesMedian, DiameterAverage, Connectivity }	2
(8)	{ BetweenneesMedian, DiameterMax, DegreeAverage }	2
(9)	{ BetweenneesMax, DiameterAverage, Connectivity, DegreeMode }	1
(10)	{ BetweenneesMix, DiameterAverage, DegreeMedian }	1
(11)	{ BetweenneesMedian, DiameterAverage, Connectivity, DegreeMedian }	1
(12)	{ BetweenneesMix, DiameterMax, DegreeAverage }	1
(13)	{ BetweenneesAverage, DiameterMax, Connectivity, DegreeMax }	2
(14)	{ BetweenneesMax, Connectivity, DegreeMedian }	1
(15)	{ BetweenneesMode, DiameterAverage, Connectivity, DegreeMix }	2
(16)	{ BetweenneesMedian, DegreeMedian }	1
(17)	{ BetweenneesAverage, DiameterMax, DegreeMode }	1
(18)	{ BetweenneesMode, DegreeAverage }	1
(19)	{ BetweenneesMedian, DiameterAverage, Connectivity, DegreeMode }	1
(20)	{ BetweenneesAverage, Connectivity, DegreeMedian }	1
(21)	{ BetweenneesMedian, DegreeAverage }	1
(22)	{ BetweenneesMax, DiameterMax, Connectivity, DegreeAverage }	1
(23)	{ BetweenneesMedian, DiameterAverage, DegreeMode }	1

Table 5.36: The graph properties and graph property combinations used in the kNN where $k = 7$ predictions that lead to the best predictions in at least one instance with nine replicas.

BC value of every replica, or in other words BC-Max is 3.5. BC-Max basically stats that replica b is by far the most important replica in the GS. Consider the RQS′ of the MCS when mapped to the GS in Figure 5.66 as shown in Table 5.2. Here the replica b is part of five of the six q_i elements of the RQS′. This shows how the BC-Max value is a good indicator for the $a_r(p)$, and the $a_w(p)$ resulting in a very successful feature in the applied kNN approach.

With GSs with more than nine replicas the accuracy of the predictions will likely decrease, but in order to train for these scenarios first a few mappings for these GSs have to be computed. As shown previously, this is currently computational to complex.

5.5.5 Conclusion

Overall it can be said that the predictions done with the kNN-approach are very good.

Especially, the BC-Max is very good at predicting the $c_r(p)$, the $c_w(p)$, the $a_r(p)$, and the $a_w(p)$.

6 The Circle Protocol

This chapter is based on the work published, by the author, in [27, 28].
Mappings have the general problem that they introduce a layer of indirection in order to use a given PNT. As shown in Section 5.4, this indirection decreases the availability of the read and write operation and/or increases their costs. Additionally, finding the best mapping has a $N!$ complexity. The Circle Protocol (CIP) was developed to use a PNT as is, without requiring an indirection. The CIP does not add any edges or vertices to the PNT. The CIP only requires the GS to be planar and connected. Figure 6.1 shows the idea behind the CIP. Two circles that

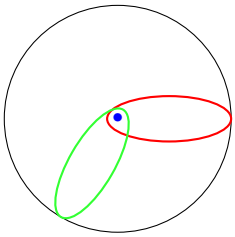


Figure 6.1: The idea behind the CIP-protocol is that two circles that embed the middle and touch the outside will always intersect.

enclosed the blue, middle dot and that touches the black, outside circle (outside) will always intersect. This is shown by the red and green circle. These circles are used to execute read and write operations. It is now a matter of projecting these structures on the vertices and edges of a GS to use them as RQs or WQs. The GS in Figure 6.2 shows such a projection. The vertices in green represent the

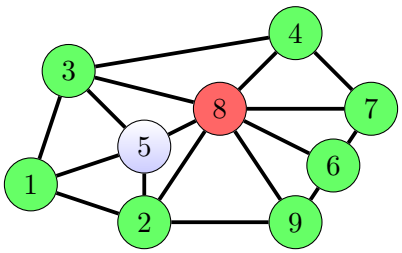


Figure 6.2: Example of the CIP.

outside. The red vertex is the so called middle. Figure 6.3 on the next page shows an orange path that touches the outside and enclosed the middle. This orange path is a circle that can be used as a quorum for either a read or write operation.

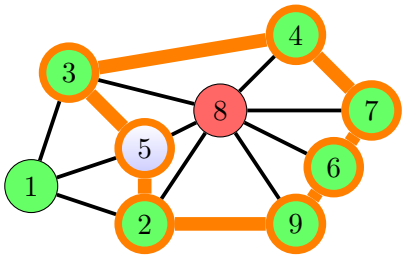


Figure 6.3: A circle that touches the outside and encloses the middle.

Any other circle that can be formed on this GS will intersect in at least one vertex with this path. Formally, a circle is a path as described in Section 2.3 on page 8 with the addition that the first and last element of the path are connected. Let m be the selected middle vertex and O a set of vertices making up the outside of a GS. Then, c is a circle that is used as a RQ or a WQ if

$$\forall o \in O \text{ there } \nexists \text{ a path } p = (m, \ldots, o) \text{ such } p \cap c \neq \emptyset \tag{6.1}$$

is true. The orange path connecting the replicas 2,3,4,5,6,7, and 9 form the

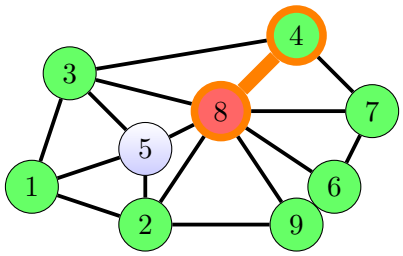


Figure 6.4: A minimized circle that touches the middle and touches the outside. The orange outlined replicas are used in the quorum. The orange edges between them represent the path through the graph to combine them.

quorum in this example. Tightening the circle around the middle and the replica 4 will eventually lead to the path shown in Figure 6.4. This process can be understood as a rubber band tightening around two points. The shortest paths, from the outside to the middle are the most cost-efficient quorum in the general case[1]. A path $q = (t, \ldots, s)$ can be used as a RQ or a WQ if t is the middle vertex and s is a vertex part of the outside. If, for example, the vertices 4 and 8 are not available, no quorum can be formed. This is because, there is no possible path available that leads around the middle and no path that leads from an outside vertex to the middle.

Previously, it was stated that it is required that the GS used is planar. To demonstrate the need for this requirement, consider the GS in Figure 6.5 on the

[1]They can be cost inefficient if the created path takes a bulk of all replicas. For instance, an inefficient path from the outside to the middle in Figure 6.4 would be 4,7,6,9,2,5,1,3,8.

facing page. In the GS in Figure 6.5 two circles can be constructed that do

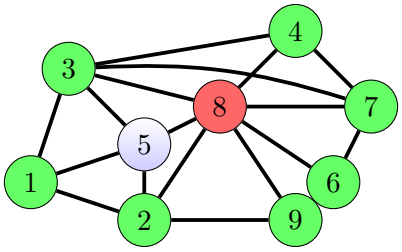


Figure 6.5: A non-planar GS used to demonstrate the need for a planar GSs when used with the CIP.

not intersect. The first circle consists of the vertices $\{4,8\}$ and the second circle considers of the vertices $\{3,7,6,9,2,5\}$. With such a GS 1SR is not given, as for example two concurrent write operation can be executed. Figure 6.6 shows a red

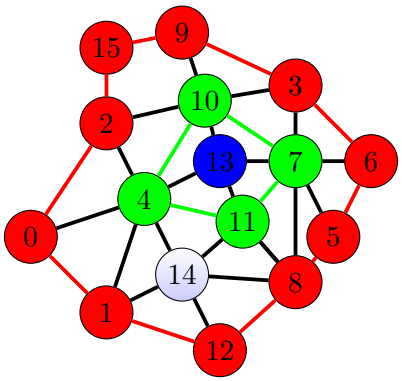


Figure 6.6: Two non intersecting circles.

and a green circle surrounding a blue middle vertex. This figure demonstrates why a circle of the CIP needs to touch the outside. If both circles were to be used as quorums, the 1SR property would not be upheld. This is because there is no intersection of the set of vertices making up the two circles.

6.1 Planarization of a Graph Structure

Previously, it was shown that non-planar graph can be used to create quorums that invalidate the 1SR criteria. As it cannot be guaranteed that the given PNT is planar and the CIP should adhere to 1SR, the PNT has to be made planar. Sometimes, a non-planar GS can be transformed into a planar GS. Consider the GS being non-planar GS in Figure 6.7 on the following page. This GS can be easily transformed into a planar GS as shown in This approach has the drawback that there is an infinite amount of possible transformations. Finding transformations that make the given GS planar and well suited for the CIP is therefore very

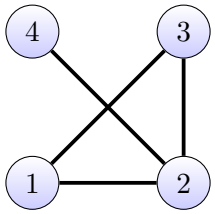


Figure 6.7: A non-planar GS.

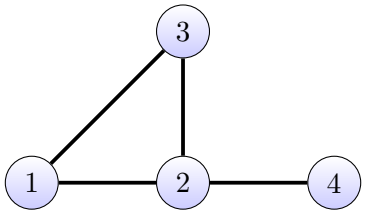
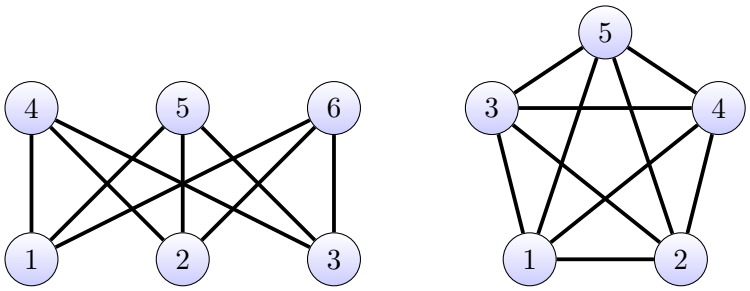


Figure 6.8: A planar GS.

complex as an infinite amount of transformation have to be tested. Additionally, not all non-planar GS can be made planar in such a way. If a subgraph of a GS is isomorph to one of the GSs shown in Figure 6.9, then the complete GS cannot be planar [29]. Another approach is to remove intersection edges for the



(a) $K_{3,3}$ (b) K_5

Figure 6.9: Two GSs that are not planar.

GS until the GS is planar. This process is shown in Figure 6.10 on the facing page. Edge removals are not considered to be not changing the PNT, as ignoring a communication link can be achieved without the need for additional hardware. Removing edges is an iterative process, as seen in Figure 6.10 and may result in multiple planar GSs. The algorithm is shown in Algorithm 10 on page 136. In line 1 in Algorithm 10 the passed GS g is put into a set of GSs. As long as this set is not empty, as shown in line 4 in Algorithm 10, the algorithm continues to process the GSs in the set. At the beginning of the `while`-loop, a GS named t is removed from the set s. If t is a planar GS, it is placed in the eventually returned set r, as shown in the `true`-branch in line 8 in Algorithm 10. If t is not planar,

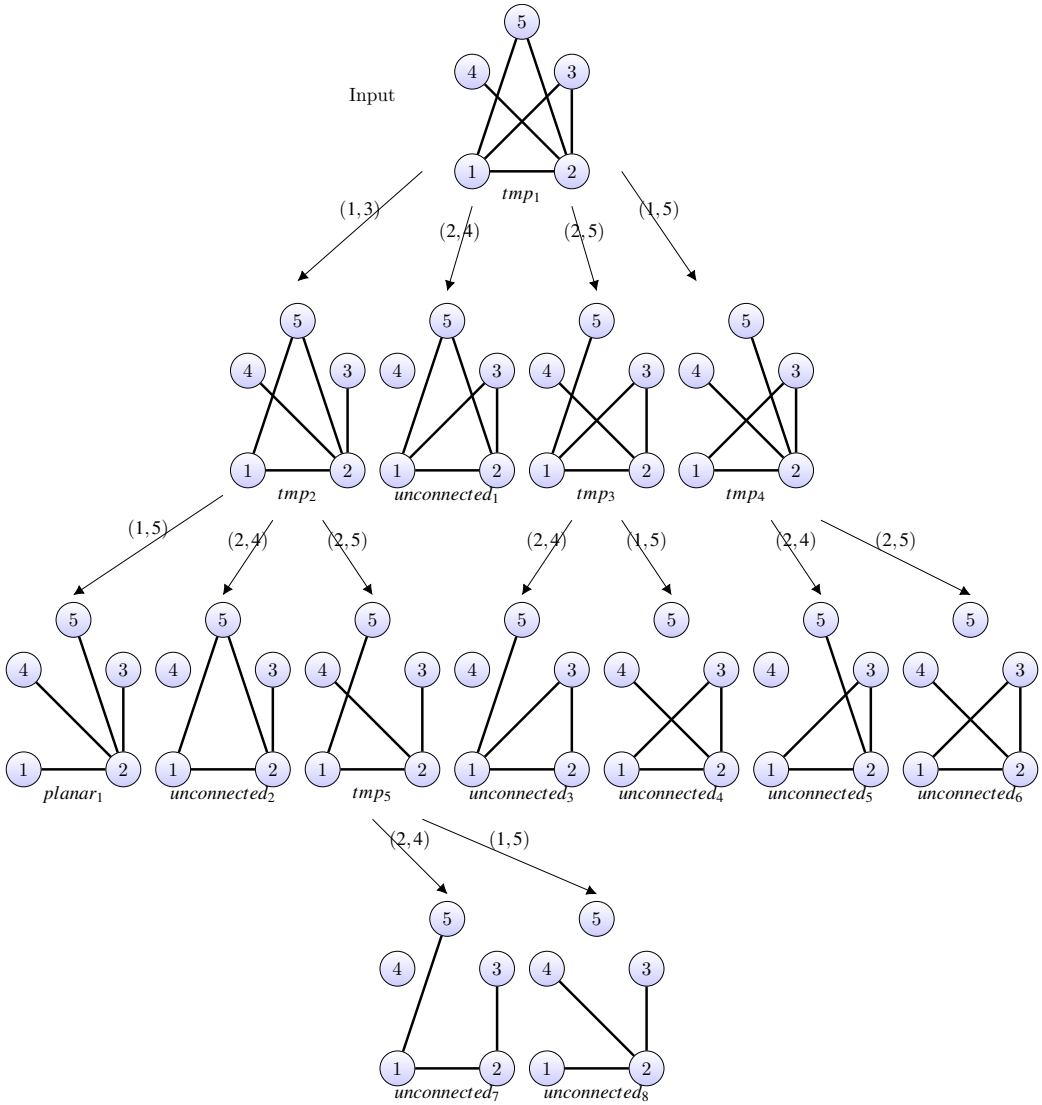


Figure 6.10: One possible transforming of a non-planar GS into a planar GSs by removing intersection edges.

Algorithm 10: Procedure *makePlanar*(g)

```
   Input: g = the GS to make planar
   Result: a set of planar GS
 1 s = {g}
 2 r = ∅
 3 d = ∅
 4 while s ≠ ∅ do
 5 |   t ∈ s
 6 |   s = s \ t
   |   /* isPlanar is a function evaluating to true if no two
   |      edges of the GS given as input intersect            */
 7 |   if isConnected(t) then
 8 |   |   if isPlanar(t) then
 9 |   |   |   r = r ∪ t
10 |   |   else
   |   |   |   /* intersectingEdges is a function returning a set of
   |   |   |      tuples of two intersecting edges of the passed GS
   |   |   |      */
11 |   |   |   forall (e_{i,j}, e_{n,m}) ∈ intersectingEdges(t) do
12 |   |   |   |   g' = (t_V, t_E \ e_{i,j})
13 |   |   |   |   g'' = (t_V, t_E \ e_{n,m})
14 |   |   |   |   if g' ∉ d then
15 |   |   |   |   |   s = s ∪ g'
16 |   |   |   |   end
17 |   |   |   |   if g'' ∉ d then
18 |   |   |   |   |   s = s ∪ g''
19 |   |   |   |   end
20 |   |   |   end
21 |   |   end
22 |   else
23 |   |   d = d ∪ t
24 |   end
25 end
26 return r
```

two intersecting edges are identified, as shown in line 11 in Algorithm 10. The GS t and these two edges are then used to create two new GSs. Each of these GSs is a copy of t minus one of the identified edges. These two new GSs are then inserted into s, as shown in line 15 in Algorithm 10.

Figure 6.10 shows an example execution of Algorithm 10. The GS labeled tmp_1 is the original non-planar GS that gets passed to the algorithm. Let t be equal to tmp_1 then $intersectingEdges(t)$ while yield the set $\{(e_{1,3}, e_{2,4}), (e_{1,3}, e_{2,5}), (e_{1,5}, e_{2,4})\}$. Let $e_{i,j} = e_{1,3}$ and $e_{n,m} = e_{2,4}$, then g' will equal the GS labeled tmp_2 and g'' will be equal $unconnected_1$. The next pair of intersecting edges $(e_{1,3}, e_{2,5})$ only yields on the new element that gets inserted into s. This is the GS labeled tmp_3. The last pair of intersecting edges is $(e_{1,5}, e_{2,4})$. Removal of the edge $e_{1,5}$ of the GS labeled tmp_1 results in the GS labeled tmp_4. Removing edge $e_{2,4}$ does not result in a GS not already present in s. As s initially contains only one element, the original non-planar GS, the first iteration of the `while`-loop is done. Let t be tmp_2, then $(e_{1,5}, e_{2,4})$ is the set of intersection edges constructed by the call to *intersectingEdges*. Removing the edge $e_{1,5}$ results in the GS labeled $planar_1$. Removing the edge $e_{2,4}$ results in the GS labeled $unconnected_2$. Both of these GSs get inserted into the set s. In the next increment of the loop $t = unconnected_1$. This GS gets discarded, as shown in line 7 in Algorithm 10, as the GS is no longer connected. If $t = tmp_3$ then $\{(e_{1,3}, e_{2,4}), (e_{1,5}, e_{2,4})\}$ is the set of intersecting edges. Removing the edge $e_{1,3}$ results in the GS labeled tmp_5. Removing the edge $e_{2,4}$ results in the GS labeled $unconnected_3$. For the second pair of intersecting edges $(e_{1,5}, e_{2,4})$ the resulting GSs are $unconnected_4$ and $unconnected_5$. $\{(e_{1,3}, e_{2,4}), (e_{2,5}, e_{1,3})\}$ is the set of intersecting edges for the GS tmp_4. The removal of these edges results in only one new GS, which is labeled $unconnected_6$. The set s now contains the GSs $planar_1$, $unconnected_2$, $unconnected_3$, $unconnected_4$, $unconnected_5$, $unconnected_6$, and tmp_5. The GS $planar_1$ gets inserted into the result set r. All GSs labeled as *unconnected* are discarded. The remaining GS tmp_5 only has one pair of intersecting edges. Removal of these edges results in the GSs $unconnected_7$ and $unconnected_8$. These two GSs are eventually discarded and the `while`-loop in line 4 in Algorithm 10 terminates. Finally, the set of the planar GSs r is returned by Algorithm 10.

6.2 The Outside Replicas

Defining the outside of a given GS is an important step in the CP. In order for this to work, the replicas must have static positions. This means that the position vector of each replica must have the same dimension as the dimension in the given topology. Two-dimensional topologies are easy to present in a two-dimensional medium and, more importantly, can be found in nearly all real-world networks.

The trivial cases are GSs with zero and one vertex. For zero vertices the outside is empty, for one vertex it is that one vertex.

That means the CIP will not work on such a GS as the middle vertex must not part of the outside. For one replica, the outside is this one replica. Again, the

Algorithm 11: Procedure $angle(E_{a,b}, E_{b,c})$

Input: $E_{a,b}$ = the first edge
Result: the angle between the two edges

1 $E_{b,c}$ = the second edge
2 $A = (b.x - a.x, b.y - a.y)$
3 $O = (c.x - b.x, c.y - b.y)$
4 $dividend = a.x * o.x + a.y * o.y$
5 $divisor = \sqrt{A_1^2 + A_2^2} \cdot \sqrt{O_1^2 + O_1^2}$
6 $tmp = acos(dividend/divisor) \cdot (180/PI)$
7 if $A_1 \cdot O_2 - A_2 \cdot O_1 < 0$ then
8 $\quad tmp = -tmp$
9 end
10 return tmp

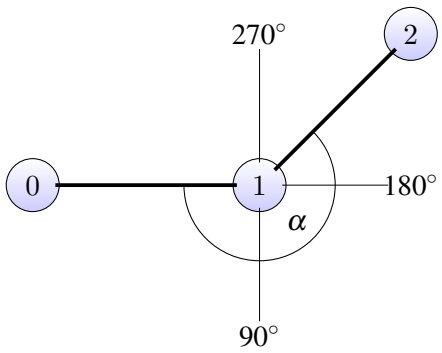


Figure 6.11: Example for the *angle*-Function described in Algorithm 11.

Algorithm 12: Procedure $nextVertex(c, j, \{adj_1, \ldots, adj_n\})$

Input: c = a vertex of the edge $e_{c,j}$
j = a vertex of the edge $e_{c,j}$
$\{adj_1, \ldots, adj_n\}$ = a set of vertices for which an edge $e_{j,adj}$ exists
Result: the vertex $adj_i \in \{adj_1, \ldots, adj_n\}$ for which the angle between $e_{c,j}$ and e_{j,adj_i} is greatest

$A := \langle adj_1, \ldots, adj_n \rangle$
$B := \langle\rangle$
$C := e_{c,j}$
for $a \in A$ do
  $B = B \cup \langle a, angle(C, e_{j,a}) \rangle$
end
$sortByAngleDecending(B)$
if $|B| > 1$ then
  return B_{2_1}
end
return B_{1_1}

Algorithm 13: Procedure $outside(g)$

Input: g = the GS to find the outside for
Result: a set of all the vertices of the outside

$s = v \in \mathbb{V}(g)$ where $min(v_x)$
$n = (s_x - 1, s_y)$
$cur = e_{n_{id}, s_{id}}$
$o = \emptyset$
while $cur \neq e_{s_{id}, n_{id}}$ do
  $e_{i,j} = cur$
  $o = o \cup e_j$
  $f = \{e_k | e_k \in e_{k,l} \in \mathbb{E}(g) \wedge e_k = e_j\}$
  $p = nextVertex(e_i, e_j, f)$
  $cur = e_{e_j, p}$
end
return o

CIP will not work on such a GS. With more than one replica the current approach takes multiple steps. The GS in Figure 6.12 is used as an example to demonstrate the algorithm shown in Algorithm 13 on the previous page. The first step is to find the leftmost vertex of the GS, as shown in line 1 in Algorithm 13. The leftmost vertex has the ID 1 as shown in Figure 6.13. The next step is to create a new edge called *cur* in the algorithm, short for current. The lines 2 and 3 show the construction of this edge in Algorithm 13. The variable o declared on the next line stores the vertices that are identified to be on the outside. Vertices which are elements of o are colored green in the following figures. The idea of the algorithm is to follow the adjacent edge with the largest angle. The angle between two edges is defined as illustrated in Figure 6.11 on page 138. The edge with the largest edge in relation to the current edge is considered to be the current edge of the next step in the algorithm. The loop started on line 5 in Algorithm 13 is executed until the edge *cur* is reached again. So far, it was implicitly assumed that the two edges $e_{i,j}$ and $e_{j,i}$ are equal. For the execution of Algorithm 13, it is imperative to consider these two edges to be distinct. In the current state of the execution of the algorithm edge $e_{0,1}$ is the edge *cur*. In Figure 6.14 on the facing page, the edge *cur* is represented by the red line. Vertex 1 is inserted into the set o. The edge *cur* is adjacent to the edges $e_{1,2}$, $e_{1,5}$, $e_{1,3}$, and $e_{1,0}$. On line 8 in Algorithm 13, a set f is constructed from these edges. During this execution of the `while`-loop f contains

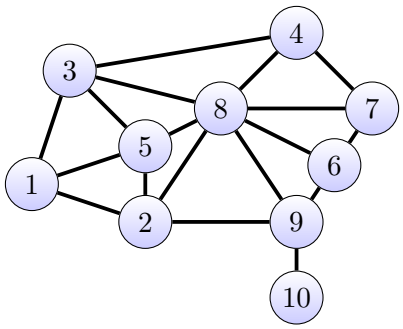


Figure 6.12: The GS, the outside replicas should be found for.

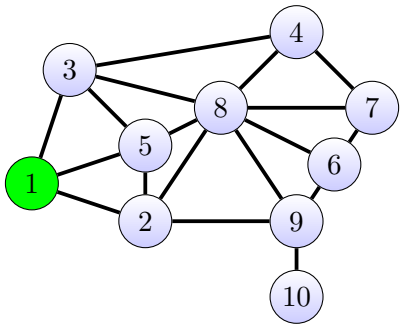


Figure 6.13: The GS the outside replicas, after step 1.

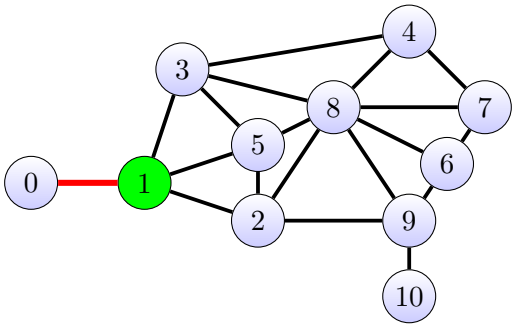


Figure 6.14: The GS the outside replicas, after step 2.

the vertices $\{2,5,3,0\}$. The vertices of the edge *cur* as well as the set f are then

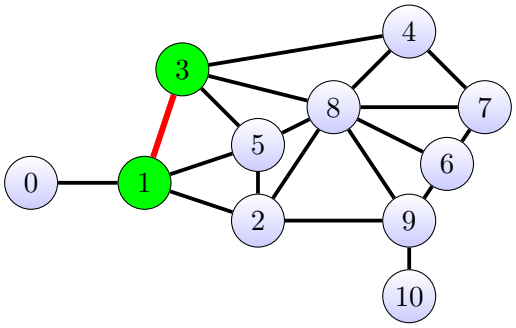


Figure 6.15: The GS the outside replicas, after step 3.

passed to Algorithm 12 on page 139. Algorithm 12 determines the adjacent edge with the largest angle. Even more specifically, Algorithm 12 determines the vertex $v \in f$ which is part of an edge $e_{z,v}$ where $e_{q,z}$ is the current edge *cur*.

The algorithm *nextVertec* used the algorithm *angle* to calculate the angle between all edges. Interestingly, given an edge $e_{i,j}$, the adjacent edge $e_{j,i}$ always has an angle of 360°, the largest possible angle. Intuitively, *nextVertex* would therefore always i to be the next vertex. This would always, erroneously, terminate the whole algorithm after the first iteration of the `while`-loop of the *outside* algorithm. To circumvent this problem, line 8 in Algorithm 12 is required. This part of the algorithm makes sure to only return this vertex if there is no other adjacent edge. If there is another edge, the vertex being part of the edge with the second largest angle, is returned. This leads to the vertex with ID 3 being returned and used in the construction of the edge *cur* as shown in line 10 in Algorithm 13. Figure 6.15 shows the state after step 3 of the algorithm when the loop-body begins execution again. The edge *cur* is again marked in red. Vertex 3 is connected to four vertices $\{1,5,8,4\}$. Passing the current edge and those four vertices to the *nextVertex* algorithm, yields the vertex with ID 4 as the next vertex. Repeating this step as before, leads to the state as shown in Figure 6.16 on the next page. The same

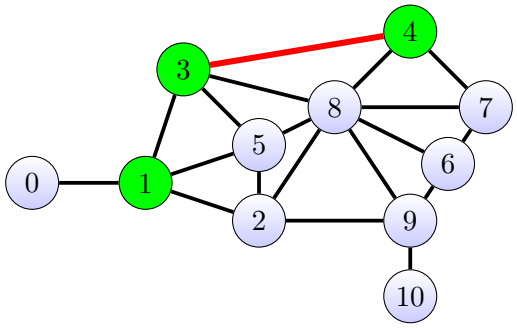


Figure 6.16: The GS the outside replicas, after step 4.

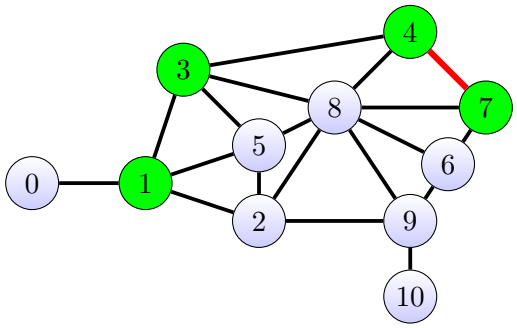


Figure 6.17: The GS the outside replicas, after step 5.

procedure is repeated for the steps shown in Figures 6.17 to 6.21. In step 9, the algorithm has reached a dead-end and has no other choice but to return through the edge he reached vertex with ID 10 in the first place. Here the `if`-branch in line 8 in Algorithm 12 is taken. The remainder of the execution of the algorithm

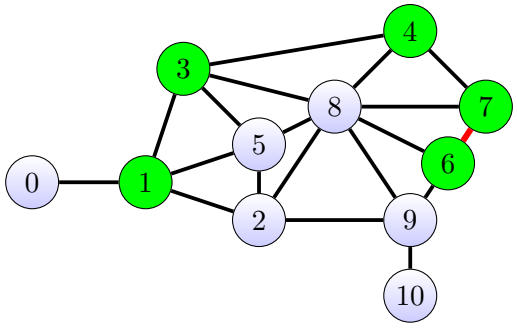


Figure 6.18: The GS the outside replicas, after step 6.

proceeds as before. This is shown in the Figures 6.22 to 6.23. Eventually, the edge $e_{1,0}$ is reached and the `while`-loop in line 5 in Algorithm 13 is left. The set o is returned, containing the vertices making up the outside of the GS.

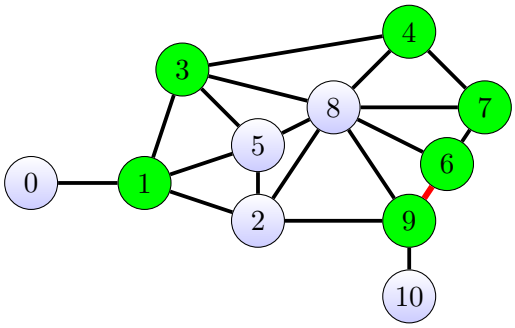


Figure 6.19: The GS the outside replicas, after step 7.

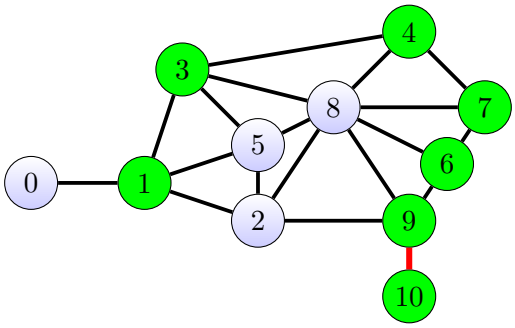


Figure 6.20: The GS the outside replicas, after step 8.

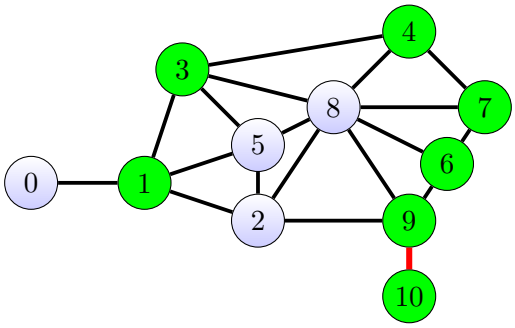


Figure 6.21: The GS the outside replicas, after step 9.

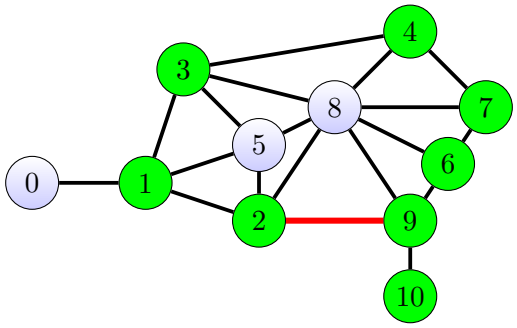


Figure 6.22: The GS the outside replicas, after step 10.

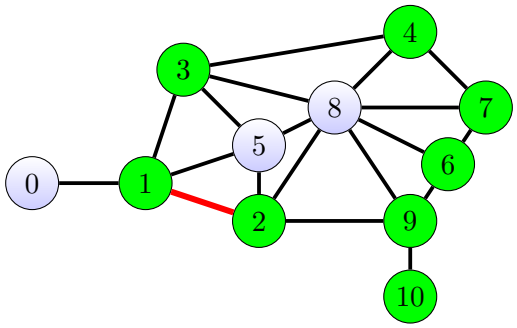


Figure 6.23: The GS the outside replicas, after step 11.

6.3 Selecting the middle

Selecting the middle has great influence on the availability and on the costs of the operations. The middle can be any vertex in the GS that is not part of the outside. Again, considering the GS of Figure 6.13 and the previously constructed outside consisting of the vertices $\{1,2,3,4,6,7,9,10\}$, vertices 5 and 8 are candidates for the middle. As the performance of the CIP depends on the GS, evaluating all possible middle vertices and comparing their performance, based on the ARW measure, guarantees to yield the best possible performance of the CIP.

Let b be the set of border vertices of a GS g, then $V(g) \setminus b$ gives the set of potential middle vertices.

6.4 Read Quorum and Write Quorum Construction

As the RQs and WQs of the CIP are constructed in the same way, the algorithm *isWriteQuorum*, shown in Algorithm 14 on the facing page, simply uses the *isReadQuorum* algorithm shown in Algorithm 15 on the next page. Both functions require the same inputs. The set of replicas that is to be tested to be a RQ or a WQ is called *replicas*. The middle is passed as *mid*. The PNT is passed with the name *pnt*. Finally, the set of outside replicas is named *outside*. The *isReadQuorum*

Algorithm 14: Procedure *isWriteQuorum*() of the CIP

Input: *replicas* = a set of replicas that is to be tested whether or not it is a RQ for the given CIP
pnt = the PNT used by the CIP
mid = the selected middle
outside = the set of replicas making up the outside
Result: true if *replicas* form a WQ, false otherwise

```
1 return isReadQuorum(replicas, pnt, mid, outside)
```

Algorithm 15: Procedure *isReadQuorum*() of the CIP

Input: *replicas* = a set of replicas that is to be tested whether or not it is a RQ for the given CIP
pnt = the PNT used by the CIP
mid = the selected middle
outside = the set of replicas making up the outside
Result: true if *replicas* form a RQ, false otherwise

```
1 shortestPathToOutside = findPathOutside(mid, outside, replicas, pnt)
2 if shortestPathToOutside ≠ ∅ then
3 |   return true
4 end
5 surroundsMiddle = testSurroundMiddle(mid, outside, replicas, pnt)
6 if surroundsMiddle ≠ ∅ then
7 |   return true
8 end
9 return false
```

algorithm shown in Algorithm 15 is straightforward. The idea of the algorithm is to either see if a path from the middle to the outside exists or if a path surrounding the middle that touches the outside exists. The first part is straight forward path computation, the second condition is done by testing the inverse property: Does a path exist that starts at the middle and reaches the outside without intersecting with the possible RQ. On line 1 in Algorithm 15, the shortest path between the middle and any of the outside replicas is constructed. If such a path can be found, *replicas* is a RQ. The procedure *findPathOutside* is presented in Algorithm 16. If no such path exists, it is tested if the middle replica is enclosed by the replicas part of *replicas*. This test is shown on line 6 in Algorithm 15. If the value returned by *testSurroundMiddle* is not the $\emptyset$, the test is considered to be successful and *replicas* is a RQ. The procedure *testSurroundMiddle* is shown in Algorithm 17 on the next page. The *findPathOutside* procedure, shown in Algorithm 16, tests

Algorithm 16: Procedure *findPathOutside*() of the CIP

```
   Input: replicas = a set of available replicas
          pnt = the PNT used by the CIP
          mid = the selected middle
          outside = the set of replicas of the outside
   Result: a smallest set of replicas that is a path from the middle to an
            outside replica
 1 if mid ∈ replicas then
 2 |   for it ∈ outside do
 3 |   |   if it ∈ replicas then
 4 |   |   |   tmp = shortestPath(mid, it, replicas, pnt)
 5 |   |   |   if tmp ≠ ∅ then
 6 |   |   |   |   return tmp
 7 |   |   |   end
 8 |   |   end
 9 |   end
10 end
11 return ∅
```

whether there is a path from the middle to an replica on the outside in the PNT that only consists of the currently available replicas. On line 1 in Algorithm 16 tests whether the middle is available. If the middle is not available, than no path to the outside starting at the middle can exist. If the middle is available, all outside replicas are iterated as shown on line 2 in Algorithm 16. On line line 3 in Algorithm 16 it is tested whether the outside replica *it* is an element of the currently available replicas. If that is the case, it is tested whether there is a path connecting *it* and the middle, as shown on line 5 in Algorithm 16. The procedure *shortestPath* finds the shortest path between two replicas in a PNT where only the available replicas/vertices are considered in the path finding procedure. If such

a path exists, it is returned from the *findPathOutside* procedure. If no path exists between the middle and an outside replica, the $\emptyset$ is returned. The procedure

Algorithm 17: Procedure *testSurroundMiddle*() of the CIP

```
   Input: replicas = a set of available replicas
          pnt = the PNT used by the CIP
          mid = the selected middle
          outside = the set of replicas of the outside
   Result: a set of replicas that encloses the middle and shares a replica
           with the outside
 1 t = {r | r ∈ pnt_V ∧ r ∉ replicas}
 2 t = t ∪ {mid}
 3 for it ∈ outside do
 4 |   s = shortestPath(mid, it, replicas, pnt)
 5 |   if s ≠ ∅ then
 6 |   |   return ∅
 7 |   end
 8 end
 9 if replicas ∩ outside = ∅ then
10 |   return ∅
11 end
12 return replicas
```

testSurroundMiddle, shown in Algorithm 17 is a complex than the *findPathOutside* procedure. As shown in line 1 in Algorithm 17 and line 2 in Algorithm 17, at first a set of replicas has to be constructed that contains all currently not available replicas plus the middle replica. As shown in the loop starting on line 3 in Algorithm 17, it is tested whether there is a path from the middle of any of the outside replicas. If that is the case, there is no enclosing circle constructed by the currently available replicas, as shown by the fact that there is a path from the middle to the outside. If no such path is found, it is finally tested whether there is an intersection between the available replicas and the outside replicas. If that is the case, a non empty set is returned. If there is no intersection the $\emptyset$ is returned.

6.5 Evaluation

The initial analyses of the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ was done on the GS shown in Figure 6.24 on the following page. For that GS, there are six possible middle vertices. These vertices are $4, 7, 10, 11, 13, 14$.

For this evaluation, the same 255 GS with eight replicas were used that were also used for the evaluation of the mapping approach. Only 20 of the 255 tested GSs with eight replicas were usable with the CIP, because the planarization approach was unable to create a planar GS. Of the 255 GSs with nine replicas only 20 were

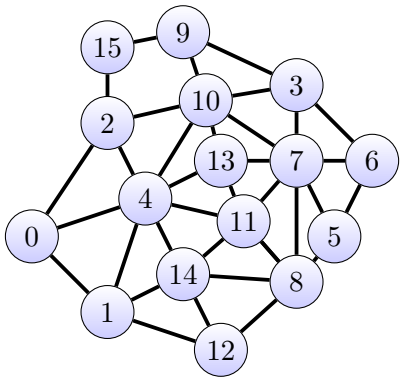


Figure 6.24: A planar GS used to do some testing with the CIP.

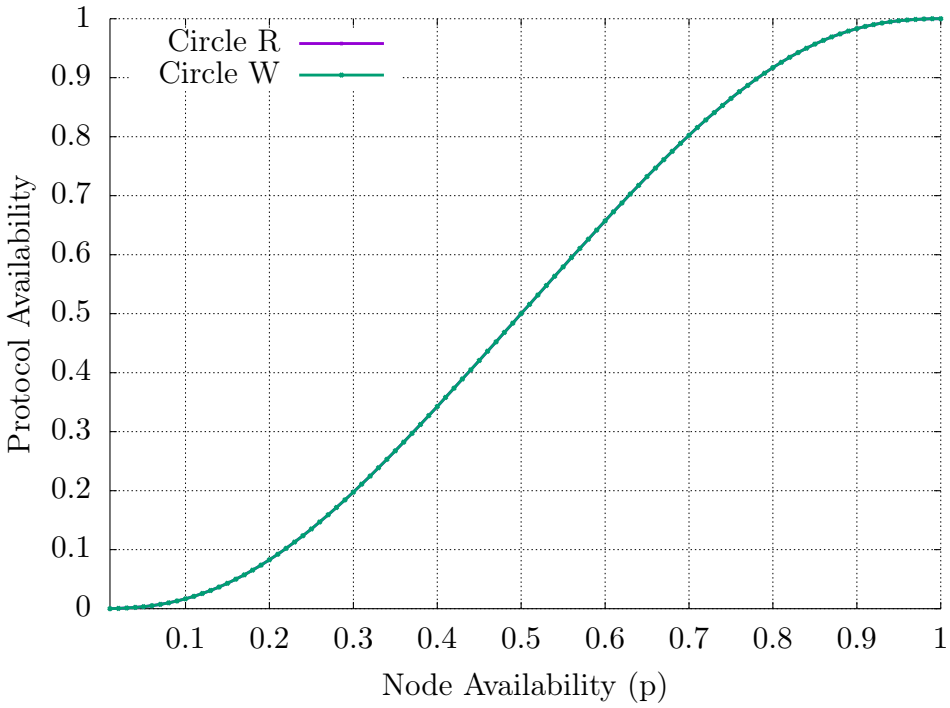


Figure 6.25: The $a_r(p)$ and the $a_w(p)$ of the CIP for the GS shown in Figure 6.24. Vertex 13 is the middle.

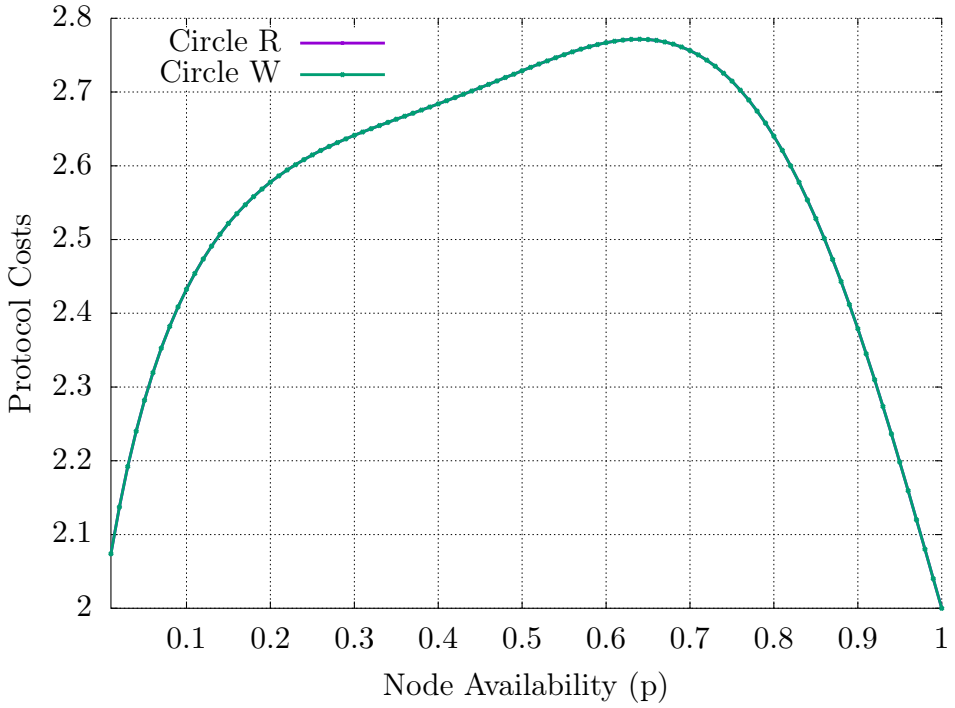


Figure 6.26: The $c_r(p)$ and the $c_w(p)$ of the CIP for the GS shown in Figure 6.24. Vertex 13 is the middle.

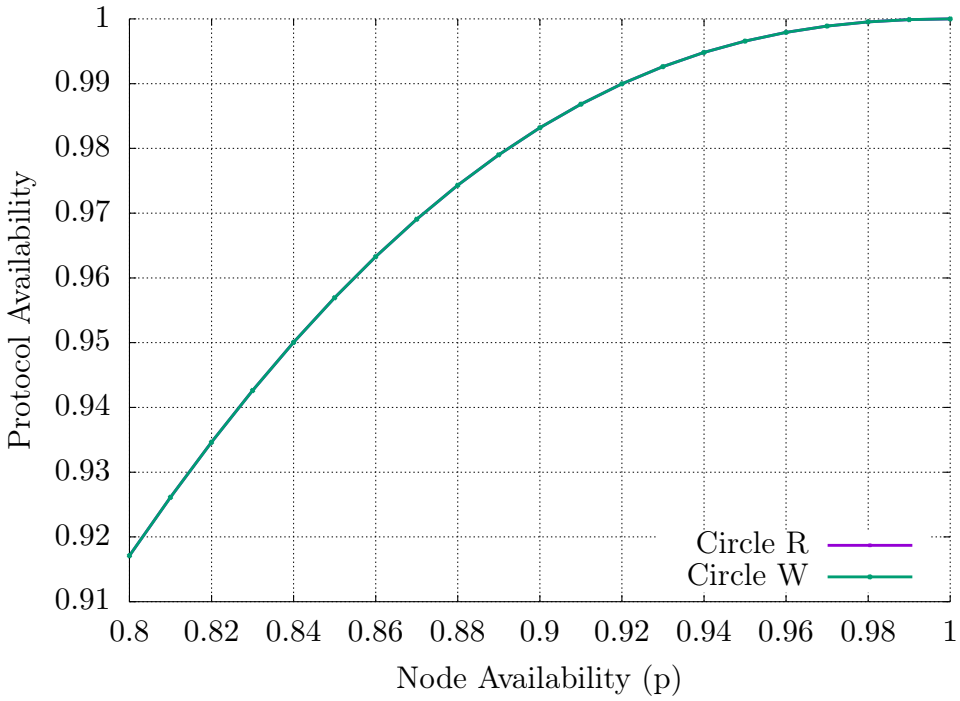


Figure 6.27: The $a_r(p)$ and the $a_w(p)$ with $p \geq 0.8$ of the CIP for the GS shown in Figure 6.24. Vertex 13 is the middle.

usable with the CIP. There are only few GSs tested because the CIP requires that the middle replica is not also part of the outside replicas. As the GSs tested are randomly generated, this is not guaranteed to be the case. For each GS all possible middle vertices where tested. Only the middle vertex with the highest ARW is included in the below shown results.

Technically, this is not so much a requirement, but a sane restriction. When the middle is part of the outside replicas $a_r(p)$ and $a_w(p)$ will always be less equal to p. This is because without the middle no outside path is going to be available. The set $\{0,1,2\}$ is the outside of the GS shown in Figure 6.28. Let replica 1 be the middle. The shortest path from the middle to the outside is therefore consists of the replicas in set $\{1\}$. If the replicas of this set are not available no quorum can be found for this GS, because otherwise the 1SR property would be violated.

Figures 6.29 to 6.30 shows the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ of the CIP on GSs with eight vertices. These are the same GSs used for the analyses of the mapping. The thick blue lines in the figures shows the non replicated $a_r(p)$ and $a_w(p)$. Of the 255 GSs tested only 20 were suitable for the CIP. The availability of the read and write operation is always below the non replicated case for $p < 0.8$. Only for $p \geq 0.8$, can the CIP converge on the non replicated operation availability, making it a bad QP for the test GS. The $c_r(p)$ and $c_w(p)$, as shown in Figure 6.30, converges on two from above. This gives insight into the topology of the tested GSs. The cheapest quorum consists of two replicas. This can only be a path from the middle to an outside replica. This is a big disadvantage for the CIP, because if these two replicas are not available no operation is possible.

A similar situation is shown for the 255 GSs with nine replicas previously tested with the mapping approach. Here the CIP converges faster towards the non replicated operation availability, but still never exceeds it.

For these 310 GSs, there is no value in applying the CIP, especially in comparison with mapped QPs.

Due to the limited number of test cases, the amount of tested GSs was expanded. The Figures 6.33 to 6.36 shows the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ of the CIP on GSs with eight and nine vertices For this test, 50000 GSs with eight and 50000 GSs with nine vertices were created. For the eight vertices GSs, 3547 were usable by the CIP. The operation availability was significantly higher in comparison to the previous tests, but still not as good as some of the mapped GPs. These results indicate that choosing an existing QP and mapping it a given PNT will likely yield better results than the CIP.

The operation availability improved further for the GSs with nine replicas, but still not reaching the mapped QPs.

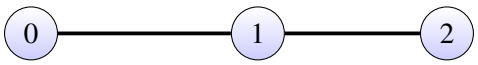


Figure 6.28: An example why the middle replica should not also be an outside replica.

Figure 6.37 on page 161 shows the operation availability of the CIP for GSs with ten vertices. Here only 131 of the 50000 tested GSs were usable by the CIP. These are mixed results: on the one hand it is currently not possible, in reasonable time, to find optimal mappings for other existing QP for GSs with ten vertices. On the other hand, only a fraction of the tested GSs catered to the CIP. The CIP has been shown to work for larger GS, but only a limited few. The significance of this problem is further shown in Figure 6.39 on page 163. Of the 50000 GSs tested with eleven vertices, only five were suitable for the CIP. The 50000 GSs tested of course only represent a fraction of the possible GSs with eleven vertices.

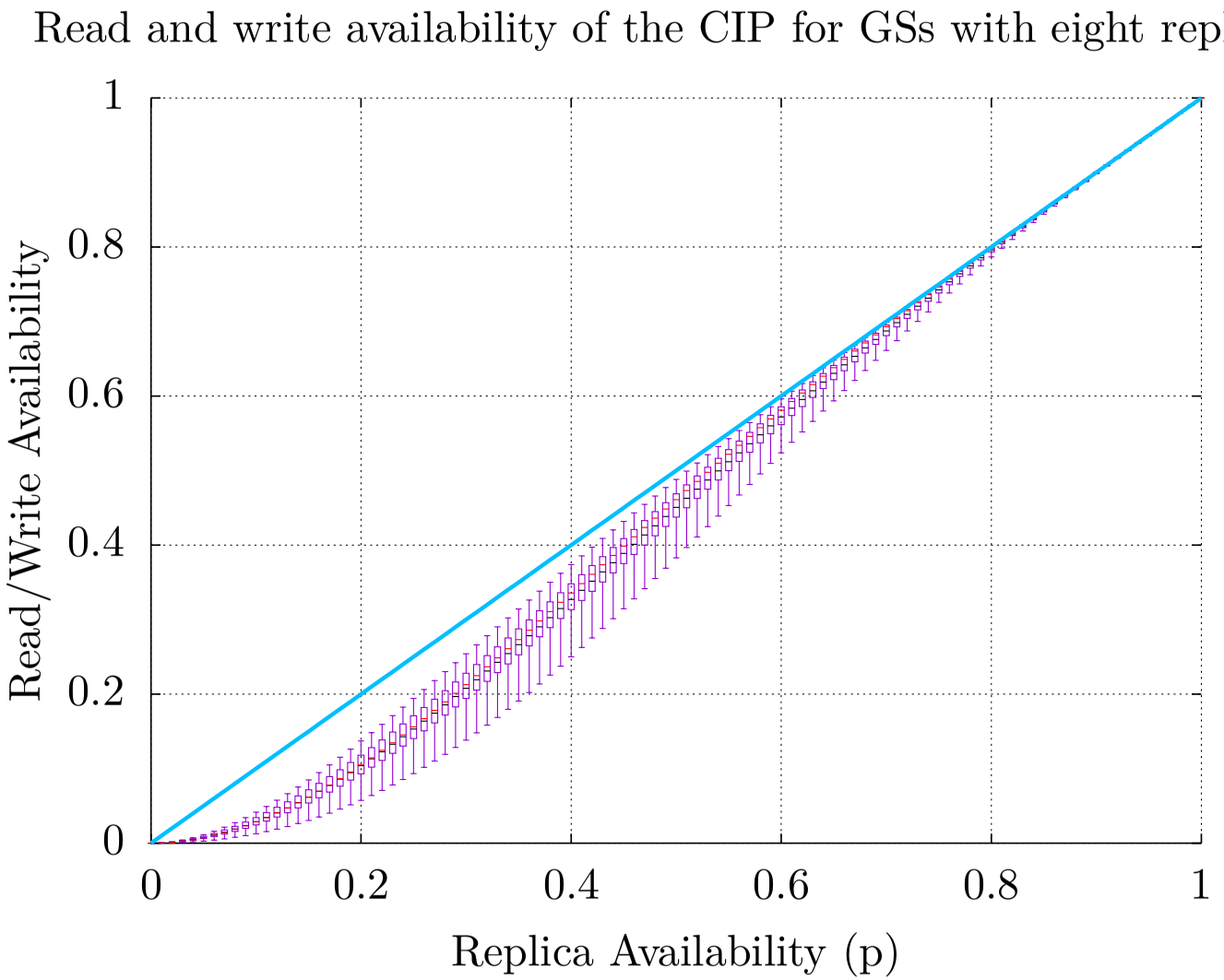


Figure 6.29: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ and the $a_w(p)$ of the CIP with eight replicas. 20 of the 255 tested simple, non-isomorphic GSs where usable by the CIP.

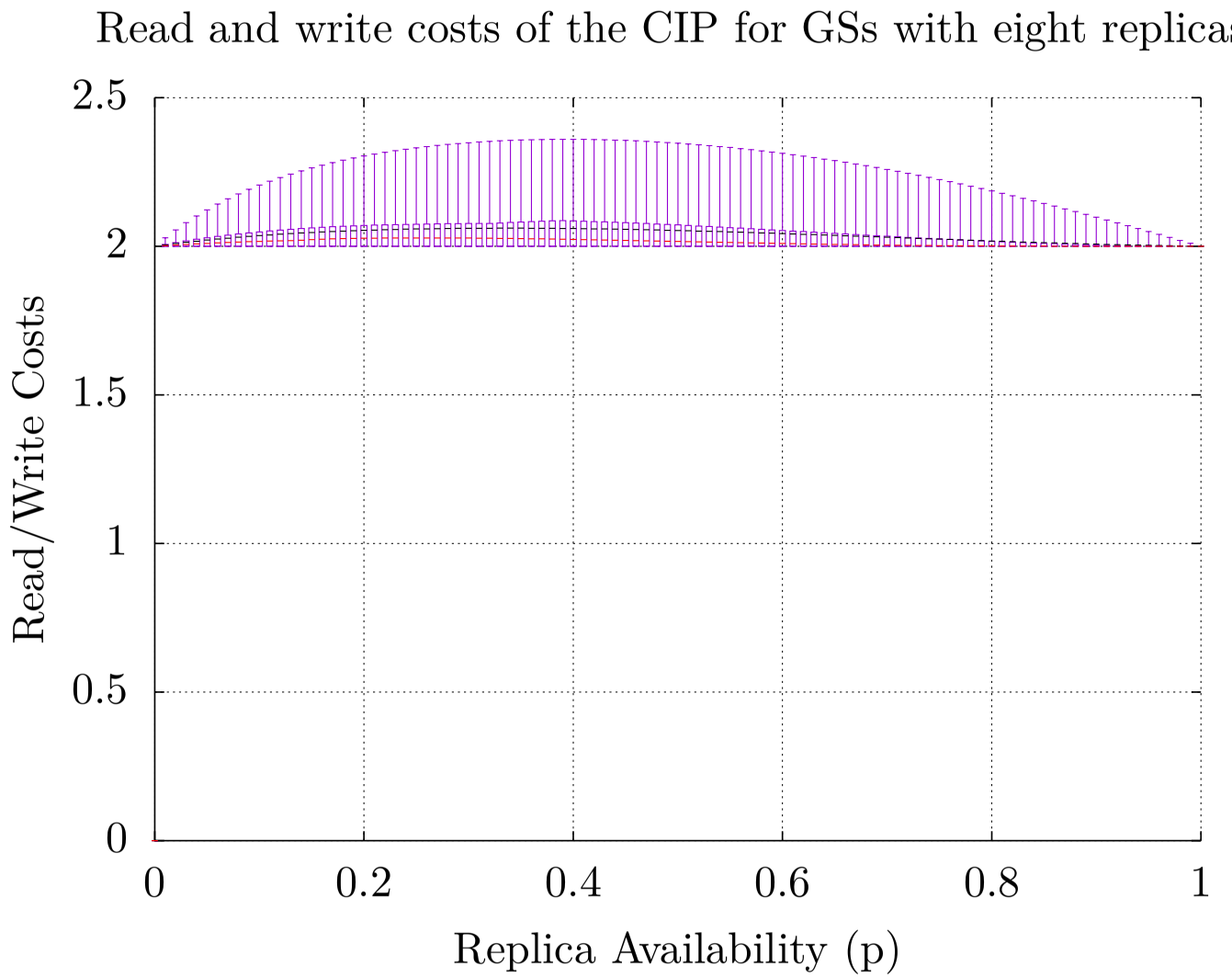


Figure 6.30: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ and the $c_w(p)$ of the CIP with eight replicas. 20 of the 255 tested simple, non-isomorphic GSs where usable by the CIP.

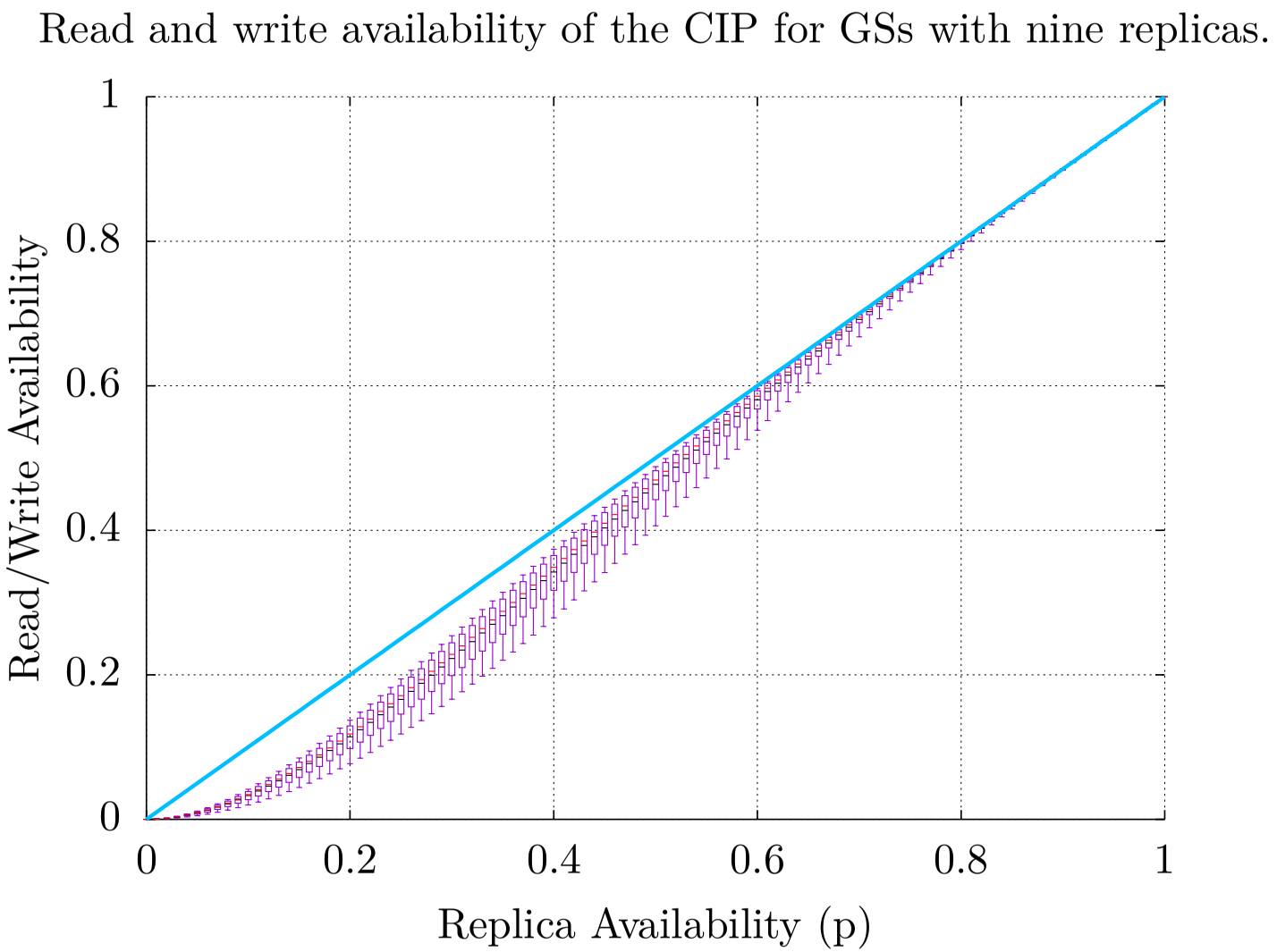


Figure 6.31: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ and the $a_w(p)$ of the CIP with nine replicas. 20 of the 255 tested simple, non-isomorphic GSs where usable by the CIP.

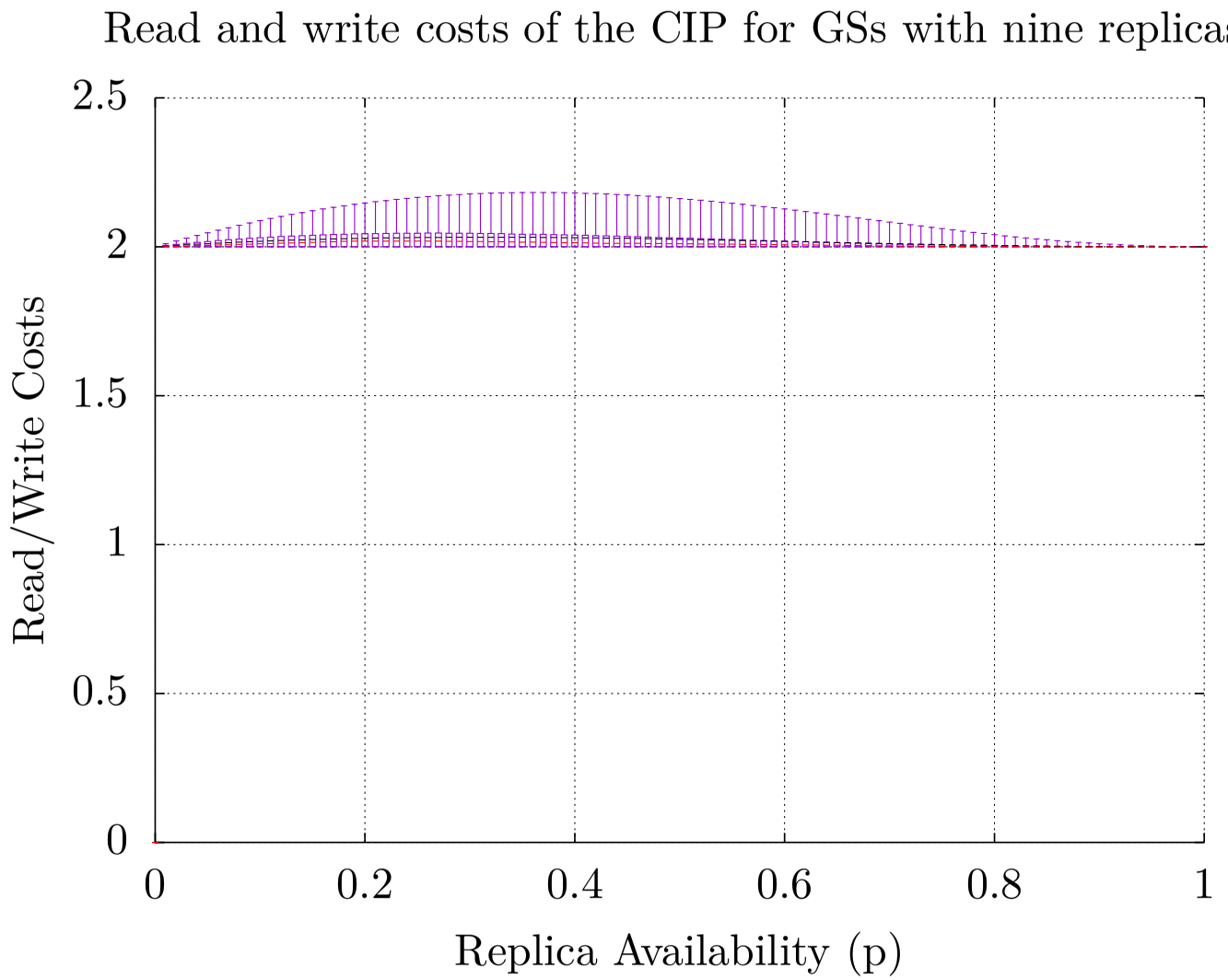


Figure 6.32: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ and the $c_w(p)$ of the CIP with nine replicas. 20 of the 2888 tested simple, non-isomorphic GSs where usable by the CIP.

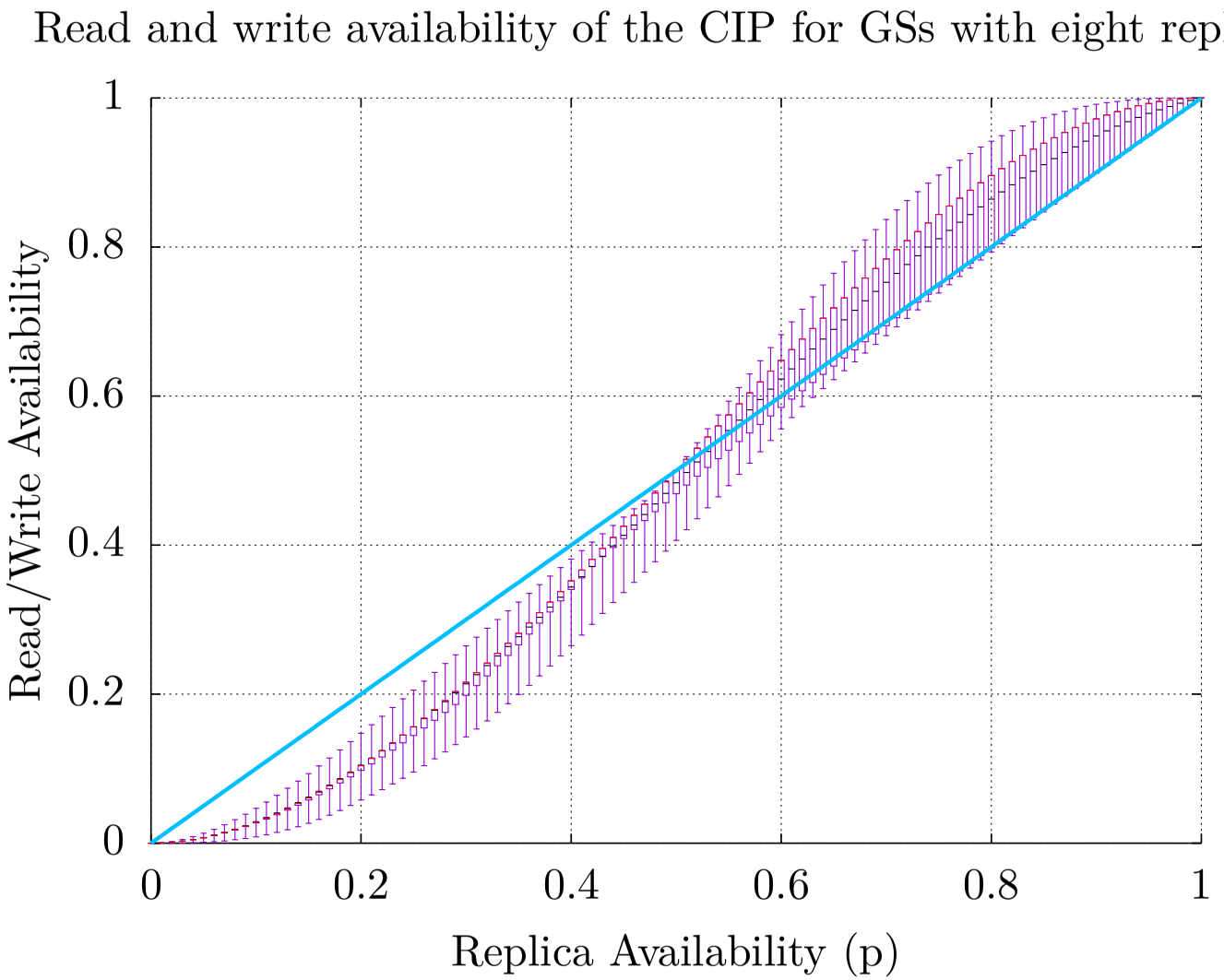


Figure 6.33: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ and the $a_w(p)$ of the CIP with eight replicas. 3547 of the 50000 tested simple, non-isomorphic GSs where usable by the CIP.

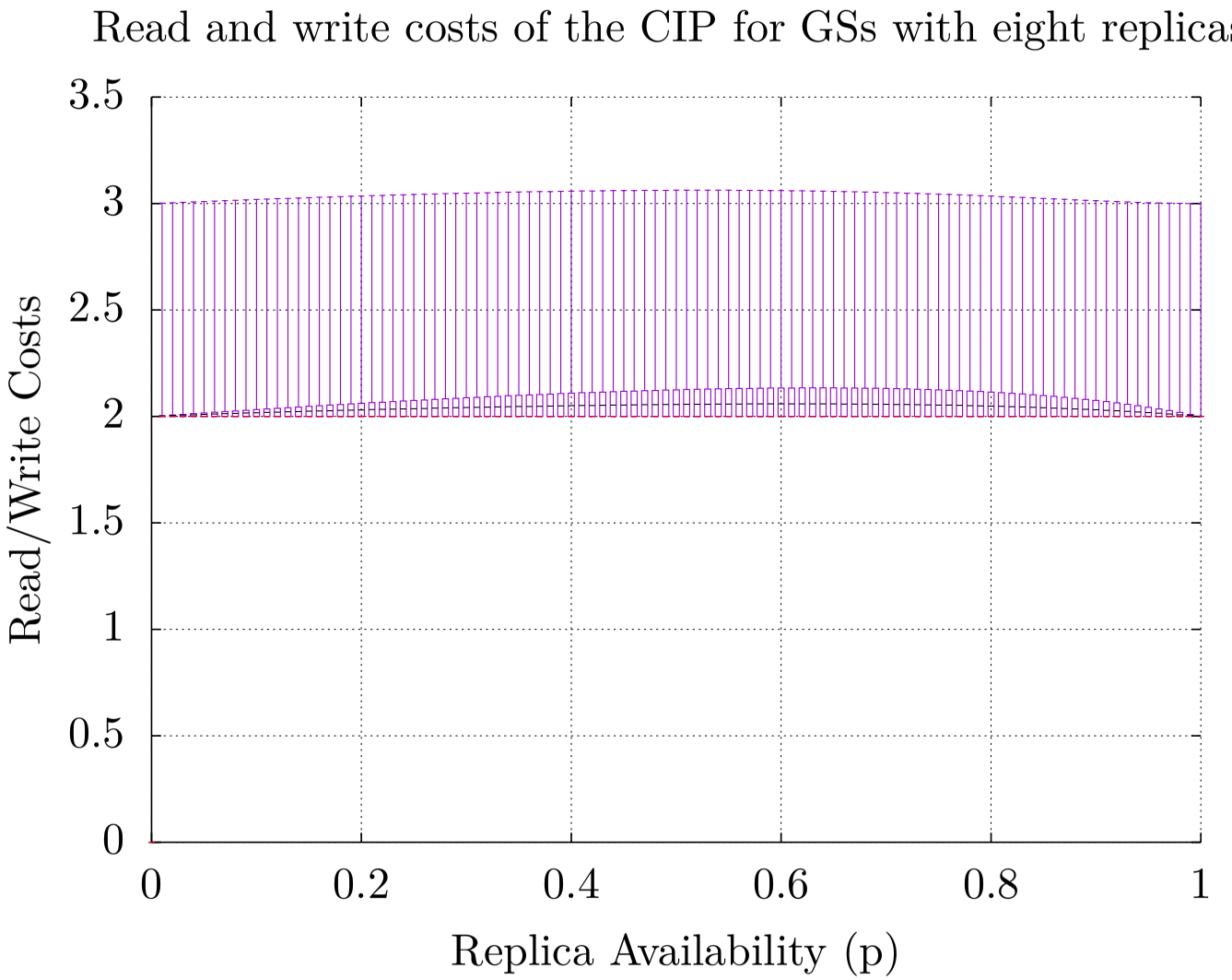


Figure 6.34: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ and the $c_w(p)$ of the CIP with eight replicas. 3547 of the 50000 tested simple, non-isomorphic GSs where usable by the CIP.

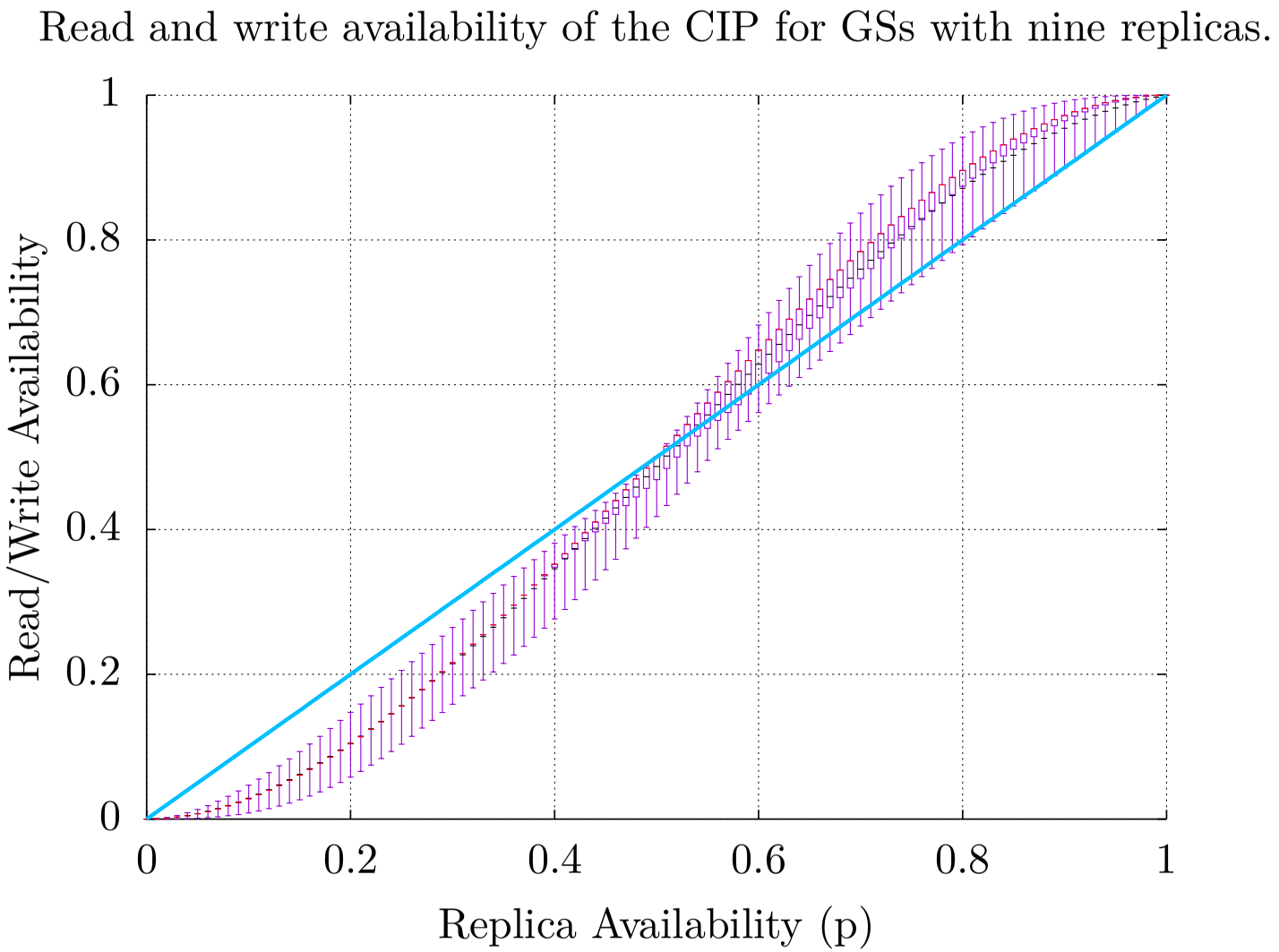


Figure 6.35: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ and the $a_w(p)$ of the CIP with nine replicas. 1052 of the 50000 tested simple, non-isomorphic GSs where usable by the CIP.

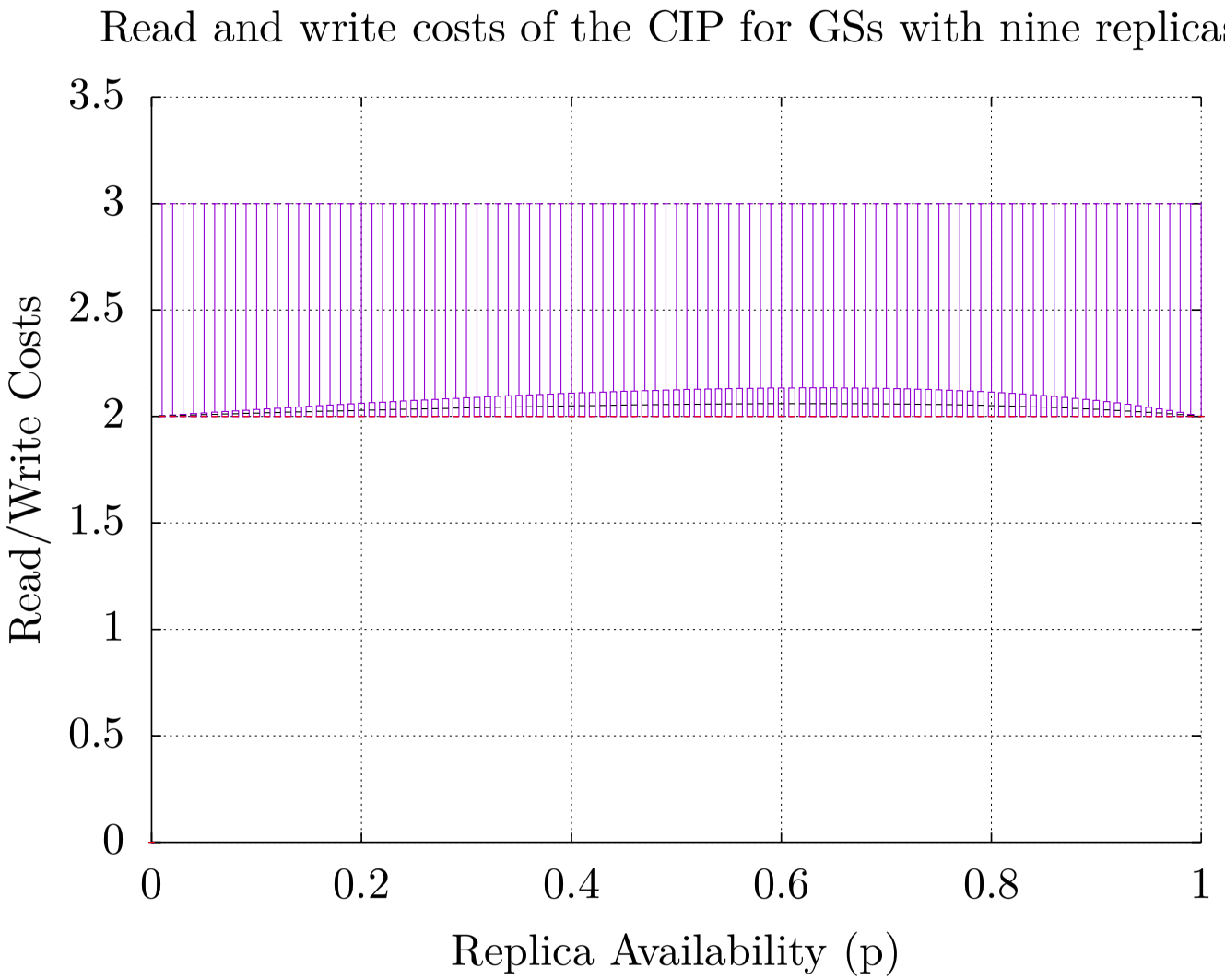


Figure 6.36: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ and the $c_w(p)$ of the CIP with nine replicas. 1052 of the 50000 tested simple, non-isomorphic GSs where usable by the CIP.

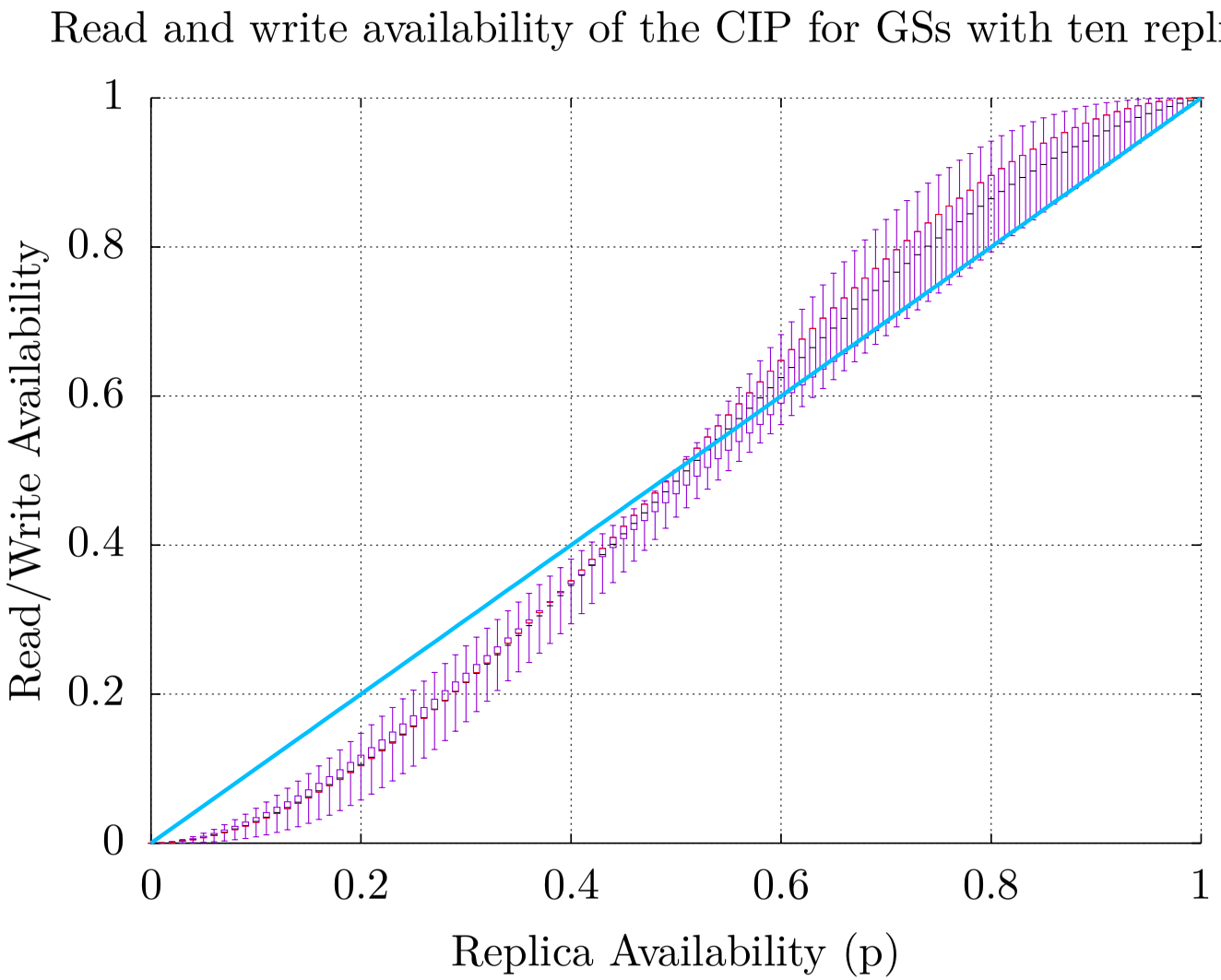


Figure 6.37: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ and the $a_w(p)$ of the CIP with ten replicas. 131 of the 50000 tested simple, non-isomorphic GSs where usable by the CIP.

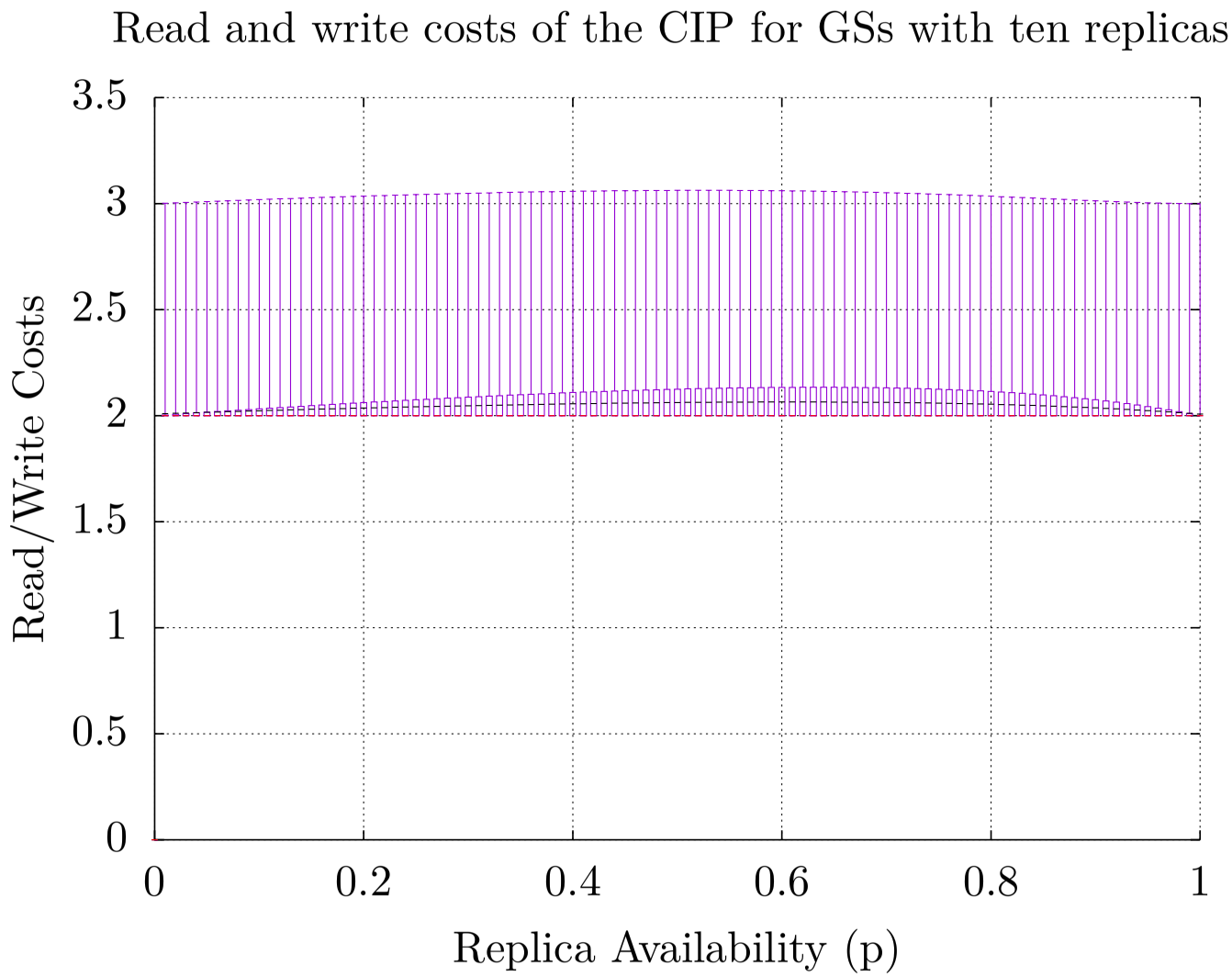


Figure 6.38: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ and the $c_w(p)$ of the CIP with ten replicas. 131 of the 50000 tested simple, non-isomorphic GSs where usable by the CIP.

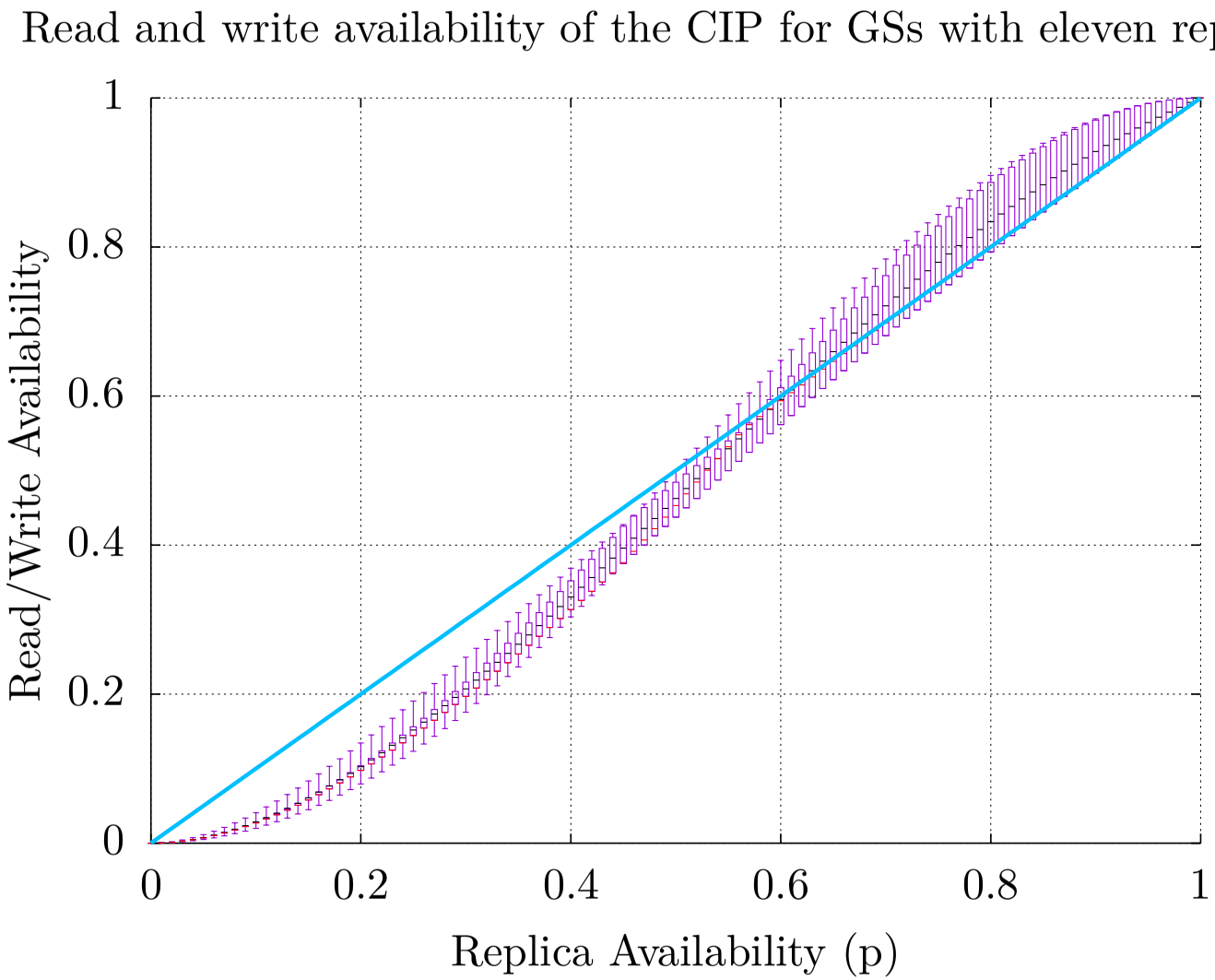


Figure 6.39: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ and the $a_w(p)$ of the CIP with eleven replicas. 5 of the 50000 tested simple, non-isomorphic GSs where usable by the CIP.

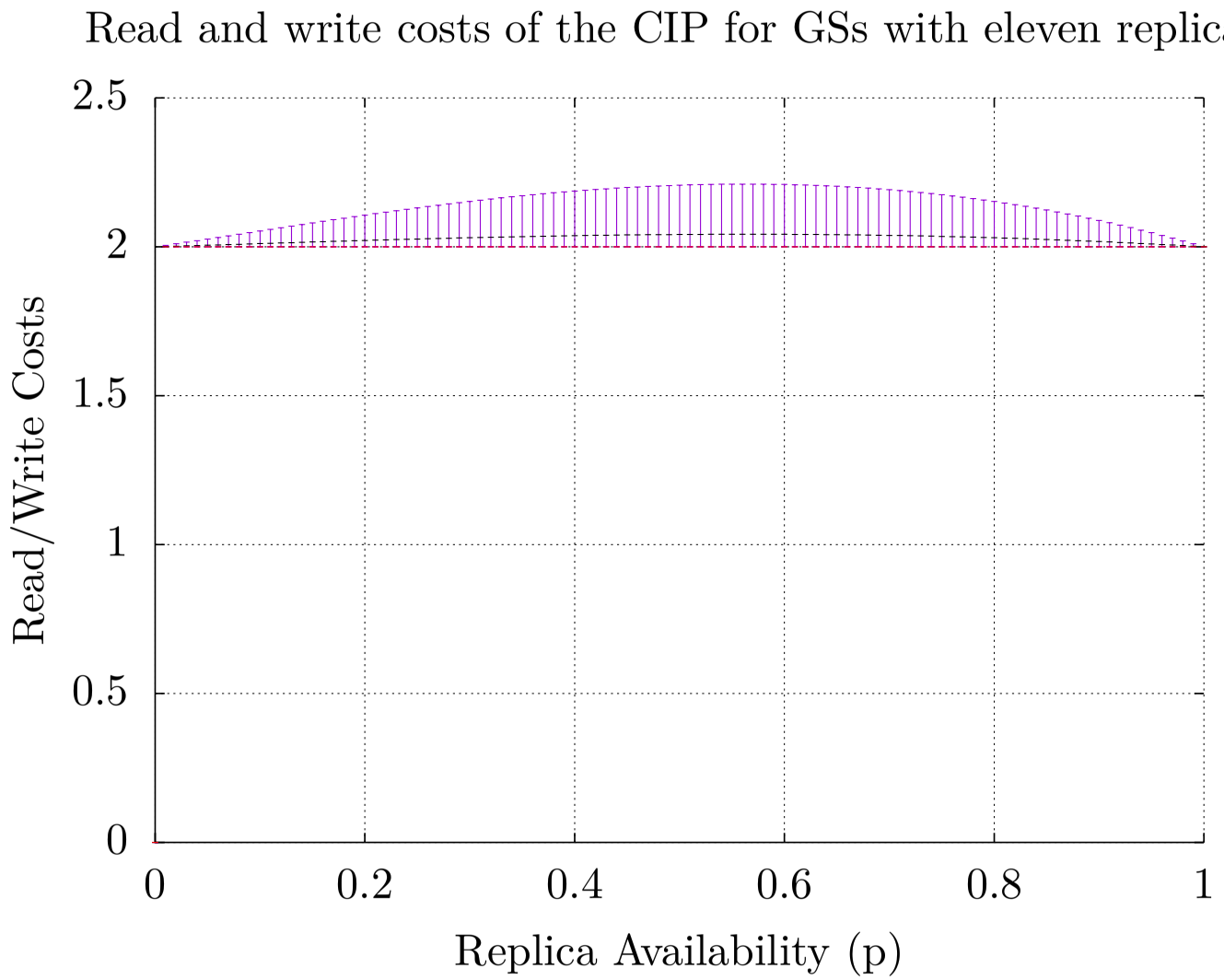


Figure 6.40: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ and the $c_w(p)$ of the CIP with eleven replicas. 5 of the 50000 tested simple, non-isomorphic GSs where usable by the CIP.

7 The Crossing Protocol

This chapter is based on the work published, by the author, in [30, 31, 28]. The Crossing Protocol (CP) can be considered to be a combination of the TLP and the CIP. From the CIP, it inherits the ability to directly work on a PNT. Actually, the first two steps of the CIP, namely making the PNT planar and selecting the outside replica, are directly carried over to the CP. From the TLP, the way of the RQs and the WQ construction is borrowed. The CP takes those two assets and combines them efficiently.

7.1 Idea and Specification

The idea behind the CP is to create intersecting paths that cross a given PNT. These paths cross the PNT vertically or horizontally. A path crosses a GS if it partitions the GS, as discussed in Section 2.3 This is heavily influenced by the TLP. The difference is that the TLP requires a particular form of GS and the CP does not.

These paths are called crossings. The vertices of the crossings are used as quorums in the CP. A WQ has to cross the PNT vertically and horizontally. A RQ has to cross the PNT vertically or horizontally. Vertical crossings must intersect with horizontal crossings. In a planar graph, every vertical crossing intersects with every horizontal crossing[1].

This results in the intersection of every RQ with every WQ, and the intersection of every WQ with every other WQ, resulting in the 1SR property. Crossings are

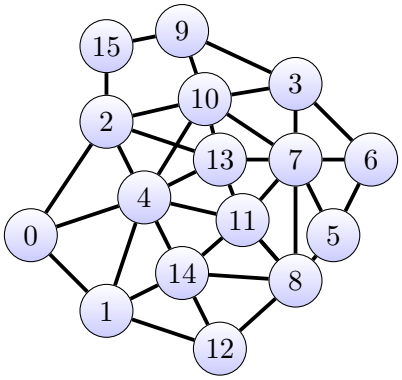


Figure 7.1: The non-planar version of the GS used to explain the CP.

[1]Compare with Figure 6.5 to see the problem with non-planar GS

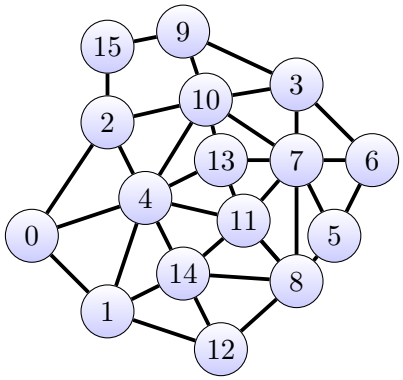


Figure 7.2: The GS used to explain the CP.

paths between specific vertices. These specific vertices lie on the outside of the GS. The outside vertices are identified in the same way as described in Section 6.2 on page 137. Figure 7.3 shows the outside vertices marked in red for the GS shown in Figure 7.2. The next step is to divide the outside vertices into four sets. These

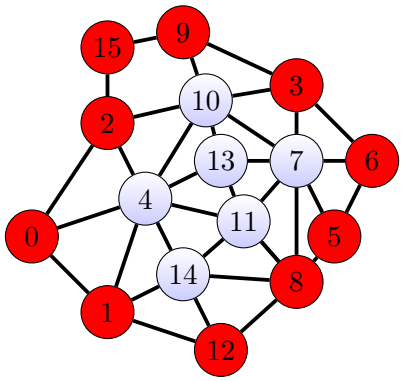


Figure 7.3: The outside replicas.

sets are called the top set (T), the bottom set (B), the left set (L) and the right set (R). Together, these four sets make up the so-called TBLR sets. All elements of the TBLR sets must be on the outside as defined in Section 6.2. Let O be the

outside. The TBLR sets are defined as follows: let U be a placeholder for any of the sets T, B, L and R.

$$U \neq \emptyset \tag{7.1}$$
$$\forall v, v' \in U : \Omega(v, v') \in U \tag{7.2}$$
$$\forall v \in O \tag{7.3}$$
$$T \cap L \neq \emptyset \tag{7.4}$$
$$T \cap R \neq \emptyset \tag{7.5}$$
$$B \cap L \neq \emptyset \tag{7.6}$$
$$B \cap R \neq \emptyset \tag{7.7}$$
$$|T \cap L| = 1 \tag{7.8}$$
$$|T \cap R| = 1 \tag{7.9}$$
$$|B \cap L| = 1 \tag{7.10}$$
$$|B \cap R| = 1 \tag{7.11}$$
$$T \cap B = \emptyset \tag{7.12}$$
$$L \cap R = \emptyset \tag{7.13}$$

As seen in the above Equation 7.4 – Equation 7.13, four pairs of outside sets intersect. These intersections are called corner-sets. The corners are named C_{TL},

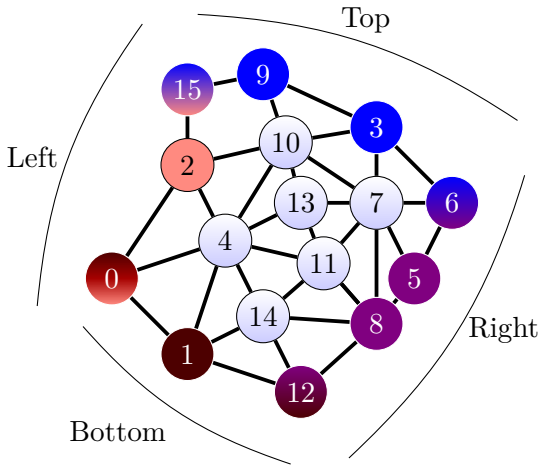


Figure 7.4: The outside replicas split into four sets.

C_{TR}, C_{BL}, C_{BR} and are defined as:

$$C_{TL} = T \cap L, \tag{7.14}$$
$$C_{TR} = T \cap R, \tag{7.15}$$
$$C_{BL} = B \cap L, \tag{7.16}$$
$$C_{BR} = B \cap R. \tag{7.17}$$

A corner C_{TL} describes the set of vertices that are part of the T and the L set. As an example for of TBLR set, consider the topology in Figure 7.4 on the preceding page. In this figure the T set consists of the vertices $\{3,6,9,15\}$. The B set consists of the vertices $\{0,1,12\}$. The L set consists of the vertices $\{0,2,15\}$. And finally the R set consists of the vertices $\{5,6,8,12\}$. The corner are $C_{TL} = 15$, $C_{TR} = 6$, $C_{BL} = 0$, $C_{BR} = 12$.

7.1.1 Definitions of Read Quorums

A RQ is a path that partitions a GS vertical or horizontal. A vertical RQ VR for a GS G is defined as:

$$\exists v, v' \in VR : v \in T \wedge v' \in B \wedge \Omega(v, v') \in G. \tag{7.18}$$

A horizontal RQ HR for a given GS G is defined as:

$$\exists v, v' \in HR : v \in L \wedge v' \in R \wedge \Omega(v, v') \in G. \tag{7.19}$$

To construct a RQ, a path in G must be found that connects the two vertices v and v'. Figure 7.5 shows a RQ. The RQ is highlighted in green and consists of the vertices $\{0,4,8,11\}$. Algorithm 18 on the facing page shows the *isReadQuorum*

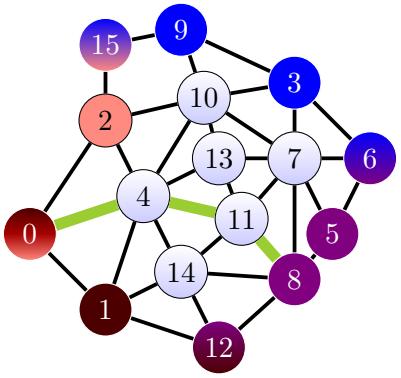


Figure 7.5: A RQ visualized by the green, thick line.

procedure of the CP. The procedure tests two properties. The first property is the existence of a path from a vertex of the T set to a vertex of the B set, which is only using the replicas in input variable *replicas*. If such a path exists, the set

Algorithm 18: Procedure *isReadQuorum*() of the CP

Input: *replicas* = a set of replicas that is to be tested whether or not it is a RQ for the given CP
pnt = the PNT used by the CP
tblr = the TBLR set of sets
Result: true if *replicas* form a WQ, false otherwise

```
1  for t ∈ tblr_t do
2  |   for b ∈ tblr_b do
3  |   |   a = shortestPath(t, b, replicas, pnt)
4  |   |   if a ≠ ∅ then
5  |   |   |   return true
6  |   |   end
7  |   end
8  end
9  for l ∈ tblr_l do
10 |   for r ∈ tblr_r do
11 |   |   a = shortestPath(l, r, replicas, pnt)
12 |   |   if a ≠ ∅ then
13 |   |   |   return true
14 |   |   end
15 |   end
16 end
17 return false
```

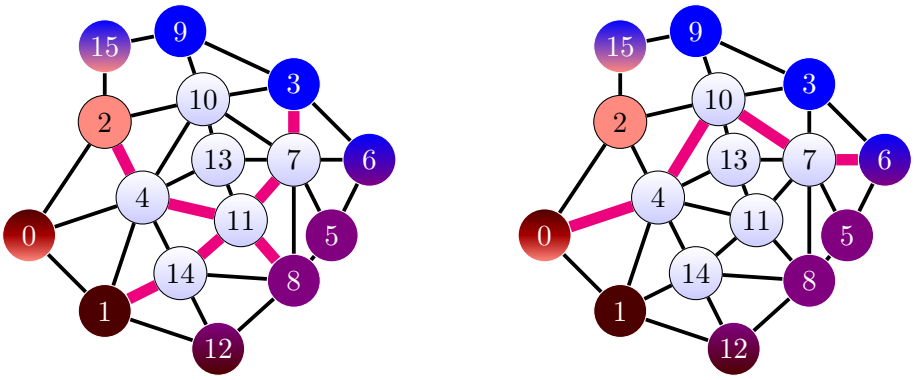
of replicas *replicas* is a RQ. The other property tests the existence of a path from a vertex of the L set to a vertex of the R set. If a path exists, the set of replicas *replicas* is also a RQ.

7.1.2 Definition of Write Quorums

A WQ is defined as:

$$QW := VR \cup HR. \tag{7.20}$$

By this definition, every union of a vertical and horizontal crossing is a WQ. Figure 7.6a shows such a WQ. The WQ is highlighted in pink and consists of the vertices $\{1,2,3,4,7,8,11,14\}$. In Section 7.1 on page 165, C_{TL}, C_{TR}, C_{BL}, and C_{BR} are defined. These corners can be used to construct a WQ. To construct a WQ from these corner-sets, a single path must be found that either connects the C_{TL} corner with the C_{BR} corner or a path that connects the C_{BL} corner with the C_{TR} corner. This single path is a WQ, as it connects all four sides, the same way a horizontal path combined with a vertical path does. Figure 7.6b shows such a WQ. The WQ is highlighted in pink and consists of the vertices $\{0,4,6,7,10\}$.



(a) A WQ visualized by the pink, thick line. (b) A WQ visualized by the pink, thick line.

Figure 7.6: WQ visualization of the CP.

In contrast to *isReadQuorum* the *isWriteQuorum* procedure, shown in Algorithm 19 on the facing page, needs to test whether *replicas* yield a horizontal and vertical path through the PNT. If a vertical path is found, it is saved in the variable *vert* as shown on line line 7 in Algorithm 19. The same is done for a horizontal path as shown on line line 15 in Algorithm 19. Finally, it is tested whether both *vert* and *hori* are not empty. Should both sets be non empty, then the replicas in *replicas* form a WQ.

Algorithm 19: Procedure *isWriteQuorum*() of the CP

Input: *replicas* = a set of replicas that is to be tested whether or not it is a WQ for the given CP
pnt = the PNT used by the CP
tblr = the TBLR set of sets
Result: true if *replicas* form a WQ, false otherwise

```
1  vert = ∅
2  hori = ∅
3  for t ∈ tblr_t do
4  |   for b ∈ tblr_b do
5  |   |   a = shortestPath(t, b, replicas, pnt)
6  |   |   if a ≠ ∅ then
7  |   |   |   vert = a
8  |   |   end
9  |   end
10 end
11 for l ∈ tblr_l do
12 |   for r ∈ tblr_r do
13 |   |   a = shortestPath(l, r, replicas, pnt)
14 |   |   if a ≠ ∅ then
15 |   |   |   hori = a
16 |   |   end
17 |   end
18 end
19 return vert ≠ ∅ ∧ hori ≠ ∅
```

7.2 Correctness Argument

To prove the correctness of the CP in regard to the 1SR property, it is shown that every vertical crossing intersects with every horizontal crossing. Furthermore, it is shown that every WQ intersects with every other WQ and every RQ.

Two crossings (respectively: "paths") $P_1, P_2 \in G$ intersect, if $G_E(P_1) \cap G_E(P_2) \neq \emptyset$. Hypothesis: Not every horizontal crossing intersects with every vertical crossing. If P_1 is a vertical crossing, then a horizontal crossing P_2 connects a vertex of both partitions V_1, V_2 created by crossing P_1. This contradicts Equation 2.6 on page 8, that states that a crossing must partition the graph. Therefore, every vertical crossing intersects with every horizontal crossing.

Every WQ consists of a vertical and a horizontal crossing. As shown earlier, every vertical and horizontal crossing intersect. Therefore, the combination of a vertical and horizontal crossing must intersect with every other WQ and every RQ.

7.3 Evaluation

To get a rough idea for the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ at first the GS shown in Figure 7.2 is analyzed. The first step is to identify the outside replicas. The outside replicas are identified by the previously described method. The result of this method is shown in Figure 7.3. The outside vertices $\{0,2,15,9,3,6,5,8,12,1\}$ can be transformed into 1680 TBLR sets. The question arises, which of these TBLR sets should be used for this GS. The ARW is used to compare TBLR sets. The TBLR set with the highest ARW is used. After testing all 1680 possible TBLR sets, the TBLR set $T = \{0,2\}$, $B = \{15,9\}$, and $L = \{2,15\}$, $R = \{9,3,6,5,8,12,1,0\}$ was identified to yield the highest ARW. Figure 7.8a on

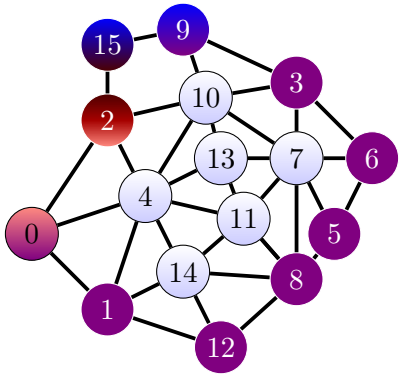


Figure 7.7: The TBLR $T = \{0,2\}$, $B = \{15,9\}$, and $L = \{2,15\}$, $R = \{9,3,6,5,8,12,1,0\}$ with the highest ARW.

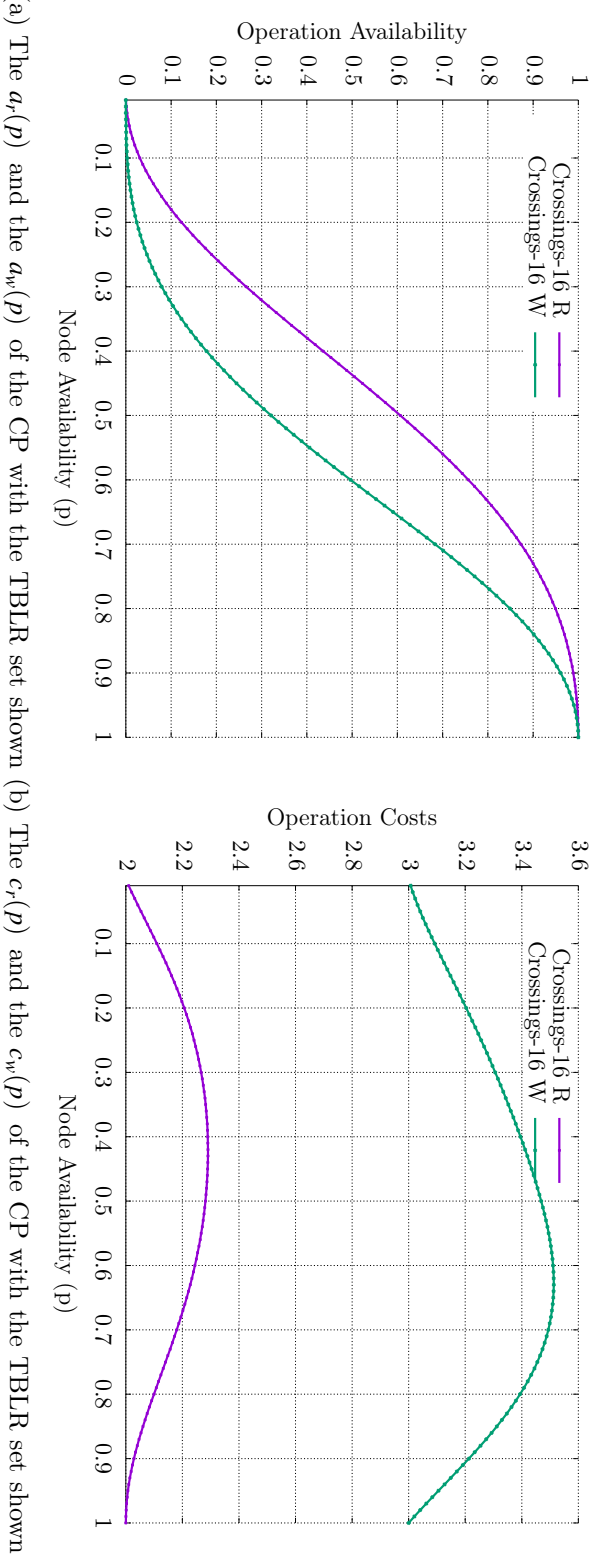
page 174 shows the $a_r(p)$ and the $a_w(p)$ of the CP for the GS shown in Figure 7.2. Figure 7.9 on page 175 shows a close-up of Figure 7.8a with $p \geq 0.8$. The $c_r(p)$ and

the $c_w(p)$ of the CP for the same GS is shown in Figure 7.8b on the following page. The availability of the read and write are worse than the operations availability of the TLP on a 4×4 GS[2], but this is expected as the GS used is not optimized for the CP. This is the point of the CP, given a random PNT it should just works. The TLP on the other hand, requires a very specific PNT. If that PNT is not available, the TLP is not available. Therefore, comparing the CP with the not mapped TLP is not worthwhile. Comparing the CP with the CIP on the other hand is much more noteworthy, as both QPs do not require any specific PNT. Given a non-planar PNT, they use the same algorithm to transform the PNT into a planar GS and then also use the same algorithm to find the outside vertices. Figure 7.10b on page 176 shows the $a_r(p)$ and the $a_w(p)$ of the CP and the CIP for the GS shown in Figure 7.2. Figure 7.10a on page 176 shows a close-up of Figure 7.10b with $p \geq 0.8$. The $c_r(p)$ and the $c_w(p)$ of the CP and the CIP for the same GS is shown Figure 7.11a on page 177. The TBLR-set of the CP is the same as in the previous example. It was calculated that the vertex with ID 11 yields the highest ARW for the CIP. Figure 7.10b shows that the $a_r(p)$ of the CP has a higher availability than the $a_r(p)$ of the CIP, but the $a_w(p)$ of the CP is lower than that of the CIP. The ARW of the CIP is 50.49 and the ARW of the CP is 69.85. The same behavior is seen for the $c_r(p)$ and the $c_w(p)$ as shown in Figure 7.10a. For high p-values ($p > 0.96$) the $a_r(p)$ and the $a_w(p)$ of the CIP are nearly identical to the $a_r(p)$ of the CP. If both, read and write operations, are equally weighted, it can be said that the CIP is better suited for the GS shown in Figure 6.24 than the CP.

Figure 7.11b on page 177 shows the $a_r(p)$ and the $a_w(p)$ of the CIP, the CP, and the TLP on a 4×4 triangular lattice as shown in Figure 4.8 on page 35. Here, the CIP is hardly better than the unreplicated case, for the most part. In the close-up, shown in Figure 7.12a on page 178, it is seen that the $a_w(p)$ of the CP catches up to the $a_w(p)$ of the CIP. With the $a_r(p)$ of the CP being higher, for the most part, than that of the CIP, it can be said that this GS is more suitable for the CP. Figure 7.12b on page 178 shows the $c_r(p)$ and the $c_w(p)$ of the three QPs. The CIP selected the vertex with the ID 6 as the middle. The ARW of the CIP is 50.49, the ARW of the CP is 71.61, and the ARW of the TLP is 73.16.

Figure 7.13 on page 180 shows the $a_r(p)$ of the CP tested with 50000 randomly generated GSs. Of those 50000 tested GSs 6169 were usable by the CP. This bad result gets somewhat relativated, considering that the CP rejects GSs where the absence of one vertex would yield the GS unusable for the QP. The bulk of the $a_r(p)$ values for those 6169 usable GSs are comparatively high. The $a_w(p)$ values, as shown in Figure 7.14 on page 181, only regularly exceed the non replicated availability with $p > 0.9$. The $c_r(p)$ are close to two as shown in Figure 7.15 on page 182. This is an indication for the structure of the GSs tested. It can be derived from this result that most GSs tested had a path that, is a RQ, only consisting of two vertices. The $c_w(p)$ on the other hand, requires between 2 and

[2] even though both GS consists of 16 vertices



(a) The $a_r(p)$ and the $a_w(p)$ of the CP with the TBLR set shown in Figure 7.7.

(b) The $c_r(p)$ and the $c_w(p)$ of the CP with the TBLR set shown in Figure 7.7.

Figure 7.8: Operation availability and cost of the CP.

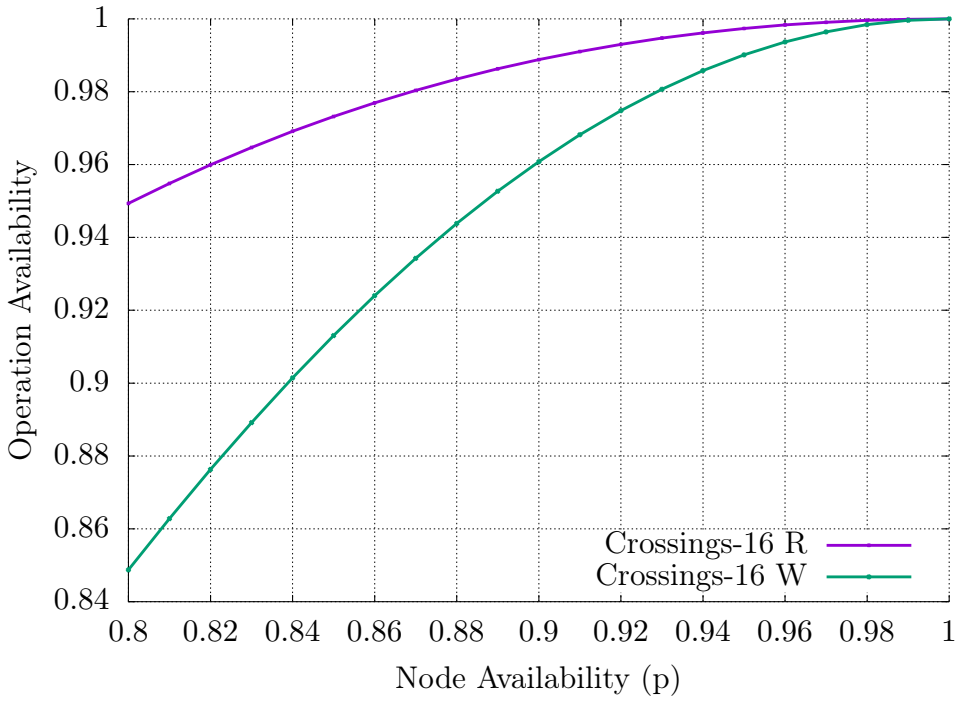
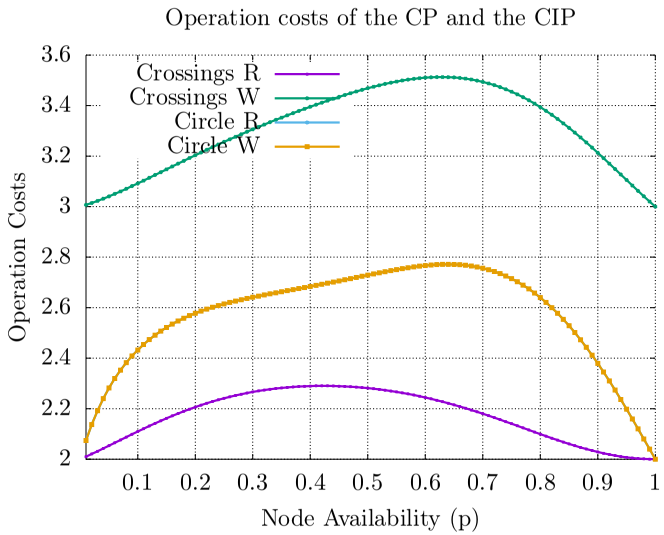
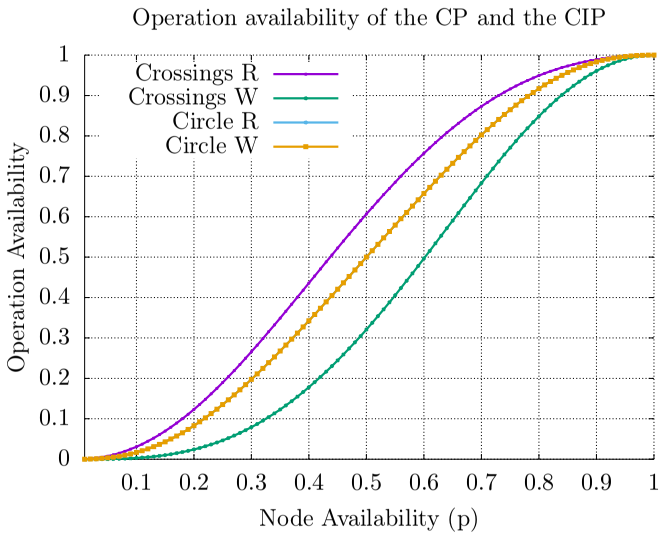


Figure 7.9: The $a_r(p)$ and the $a_w(p)$ with $p \geq 0.8$ of the CP with the TBLR set shown in Figure 7.7.

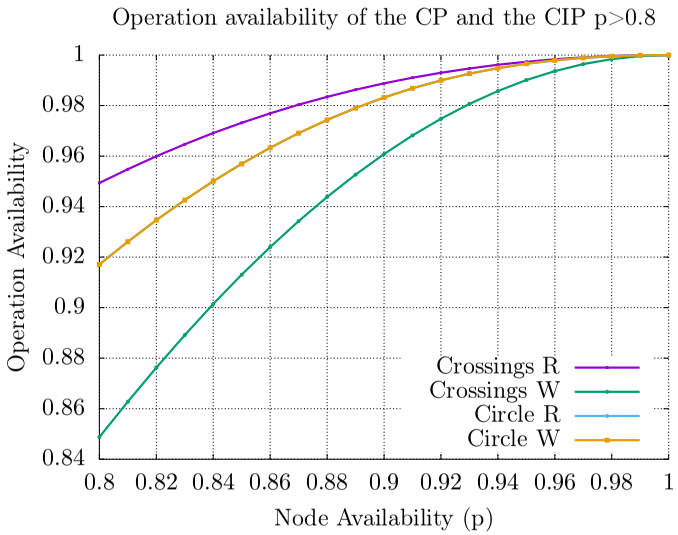


(a) The $c_r(p)$ and the $c_w(p)$ of the CP and the CIP with the TBLR set shown in Figure 7.7. The vertex 11 was used by the CIP as the middle.

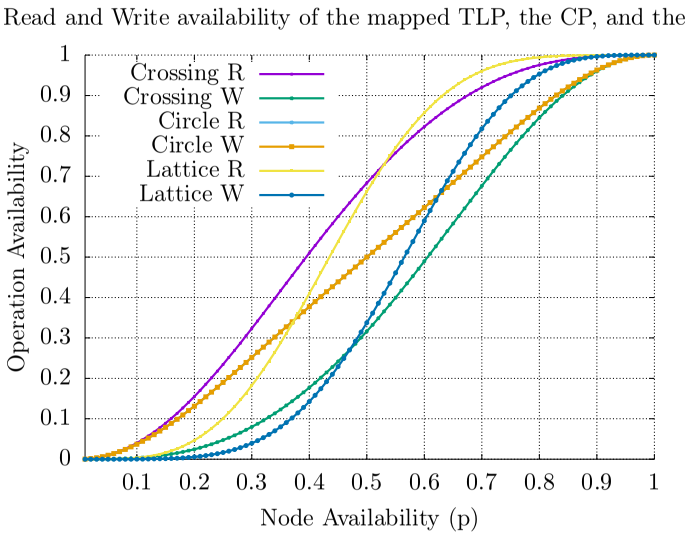


(b) The $a_r(p)$ and the $a_w(p)$ of the CP and the CIP with the TBLR set shown in Figure 7.7. The vertex 11 was used by the CIP as the middle.

Figure 7.10: Operation availability and cost comparison of the CIP and the CP.

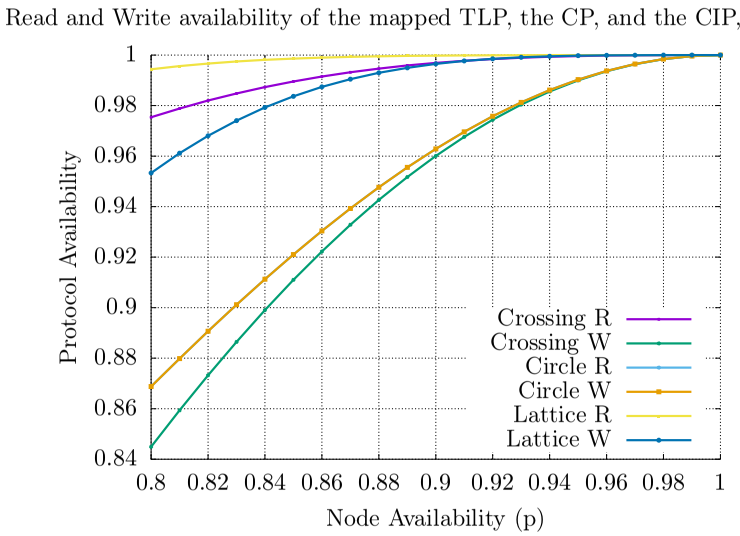


(a) The $a_r(p)$ and the $a_w(p)$ with $p \geq 0.8$ of the CP and the CIP with the TBLR set shown in Figure 7.7. The vertex 11 was used by the CIP as the middle.

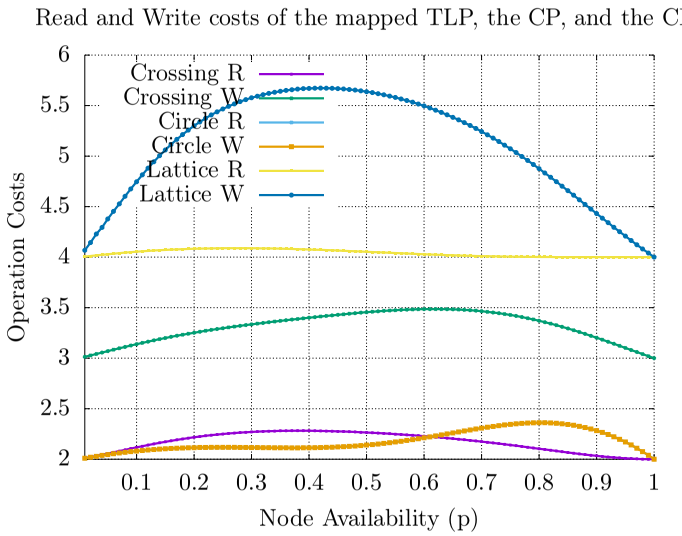


(b) The $a_r(p)$ and the $a_w(p)$ of the CP, the CIP, and the TLP. The GS used is a 4×4 triangular lattice as shown in Figure 4.8.

Figure 7.11: Operation availability comparison of the CIP, the CP, and the TLP.



(a) The $a_r(p)$ and the $a_w(p)$ of the CP, the CIP, and the TLP with $p \geq 0.8$. The GS used is a 4×4 triangular lattice as shown in Figure 4.8.



(b) The $c_r(p)$ and the $c_w(p)$ of the CP, the CIP, and the TLP. The GS used is a 4×4 triangular lattice as shown in Figure 4.8.

Figure 7.12: Operation availability and cost comparison of the CIP, the CP, and the TLP.

3.5 vertices, with the bulk between 2 and 3 vertices. Concluding, it can be said that the CP works for GSs with eight vertices.

Figure 7.16 on page 183 shows the $a_r(p)$ of the CP on GSs with nine replicas. Again, 50000 random GSs were created. Of those the CP was able to use 1555 GSs. The $a_r(p)$ for most of them is comparable with the MCS, also with nine replicas. The $a_w(p)$, as shown in Figure 7.17 on page 184, is much worse than the $a_w(p)$ of the MCS with nine replicas. This can be seen in Figure 7.18 on page 185. Only a few GSs offer a $a_w(p)$ higher than the non replicated case. The $c_r(p)$, shown in Figure 7.19 on page 186, and the $c_w(p)$, shown in Figure 7.20 on page 187, show a similar picture as with eight vertices. The $c_r(p)$ is around 2 and the majority of the $c_w(p)$ is between 2 and 3. This indicates that similar GSs, compared with the GSs with eight vertices from the previous tests, where constructed, but with nine instead of eight vertices.

The last presented results are of GSs with ten vertices. Of the 50000 GSs only 200 were usable by the CP. Figure 7.21 on page 188 shows the $a_r(p)$ and Figure 7.22 on page 189 shows the $a_w(p)$. The $a_r(p)$ is again comparable with the $a_r(p)$ of the MCS. The $a_w(p)$ on the other hand, is falling behind even more. Looking at the $c_r(p)$ and the $c_w(p)$, as shown in Figure 7.23 on page 190 and Figure 7.24 on page 191, shows why. The majority of the WQs still only consists of three vertices. But now, there are more combinations of three vertices WQs in any particular GS that can be unavailable, thus decreasing the $a_r(p)$.

Tests were conducted with 50000 GSs with eleven and 50000 GSs with twelve vertices, but the CP was unable to use any of those GSs. By checking a random selection of generated GSs it was be concluded that the problem is with the GS generation not the CP. Most graphs had many intersecting edges. After their removal often the resulting graphs approached a GS similar to a line, and lines often easily partitioned by a failure of a single vertex. After many of the GSs were made planar, only a single edge remained connecting two parts of the GS. If the outside of such a GS is constructed, often a vertex is part of both the T- and the B- or the L- and the R-set. The GS generator could be made to only generate planar GSs that were favorable to the CP, but then, the evaluation would be biased.

Overall, it can be said the performance of the CP depends heavily on the GS used. If the GS used is planar and has many edges the CP will yield good results. On the other hand, if the GS is not planar and/or poorly connected by ony few edges the CP will not work well. This is also true for mappings as shown in Section 5.4.

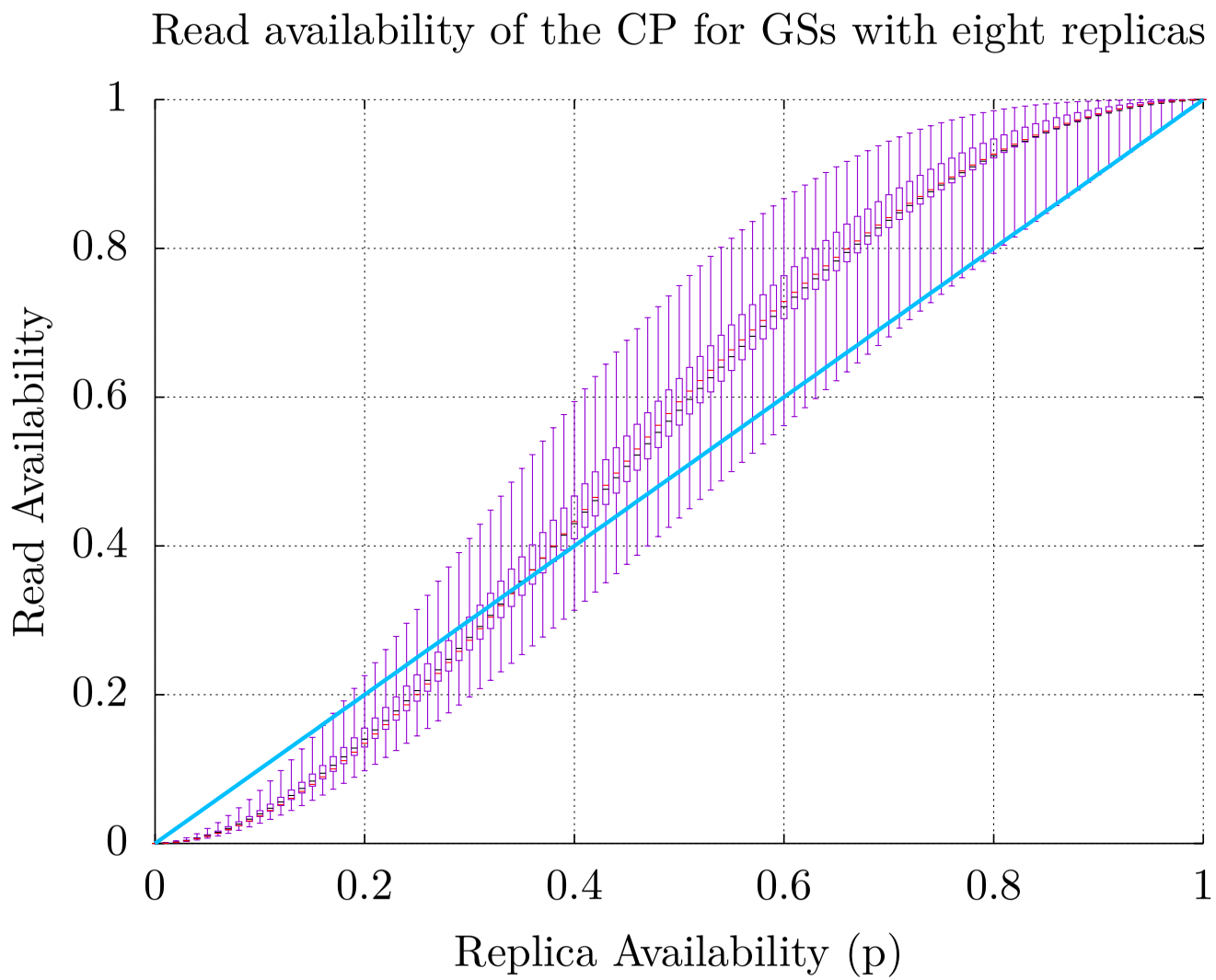


Figure 7.13: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ of the CP with eight replicas. 6169 of the 50000 tested simple, non-isomorphic GSs where usable by the CP.

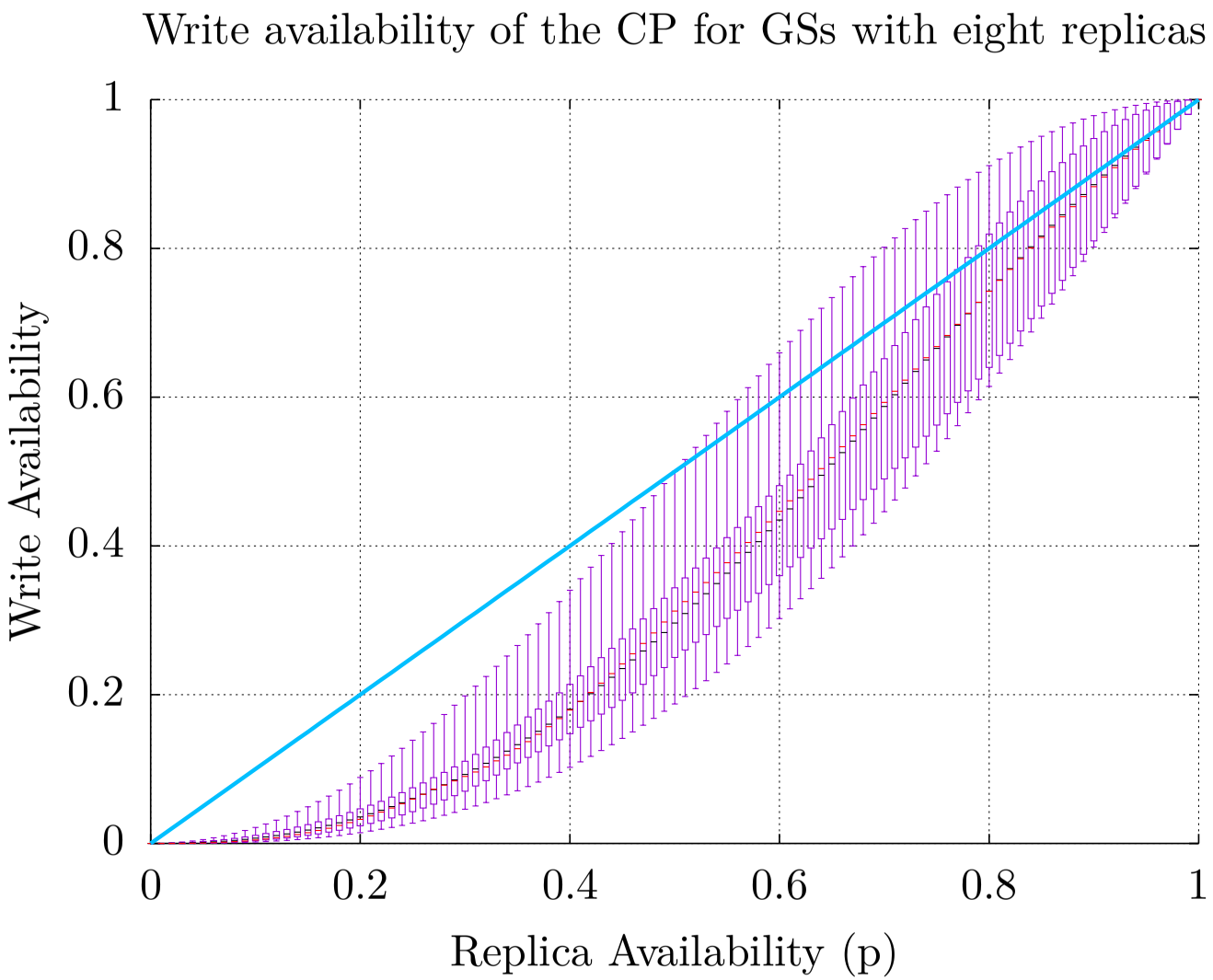


Figure 7.14: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_w(p)$ of the CP with eight replicas. 6169 of the 50000 tested simple, non-isomorphic GSs where usable by the CP.

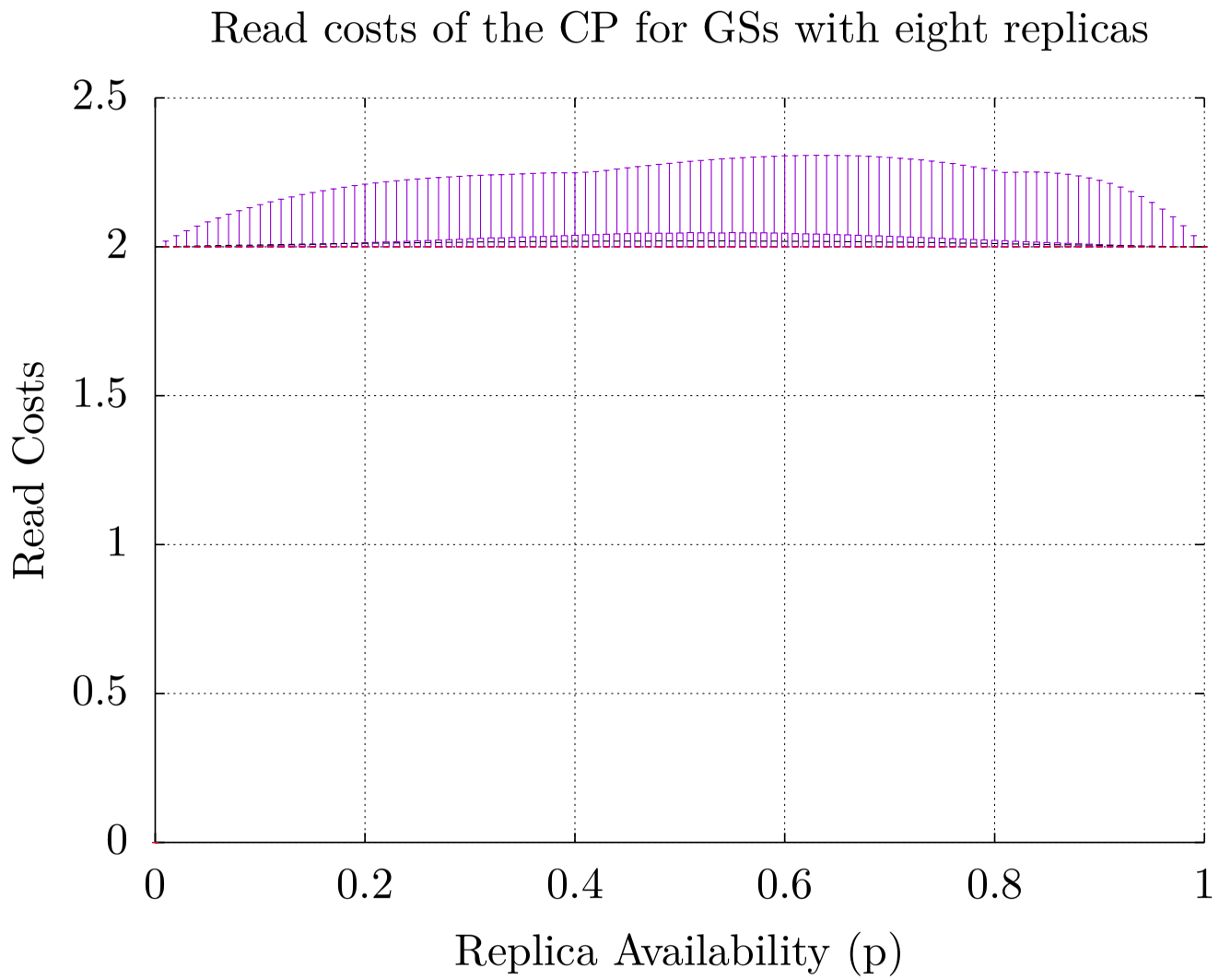


Figure 7.15: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ of the CP with eight replicas. 6169 of the 50000 tested simple, non-isomorphic GSs where usable by the CP.

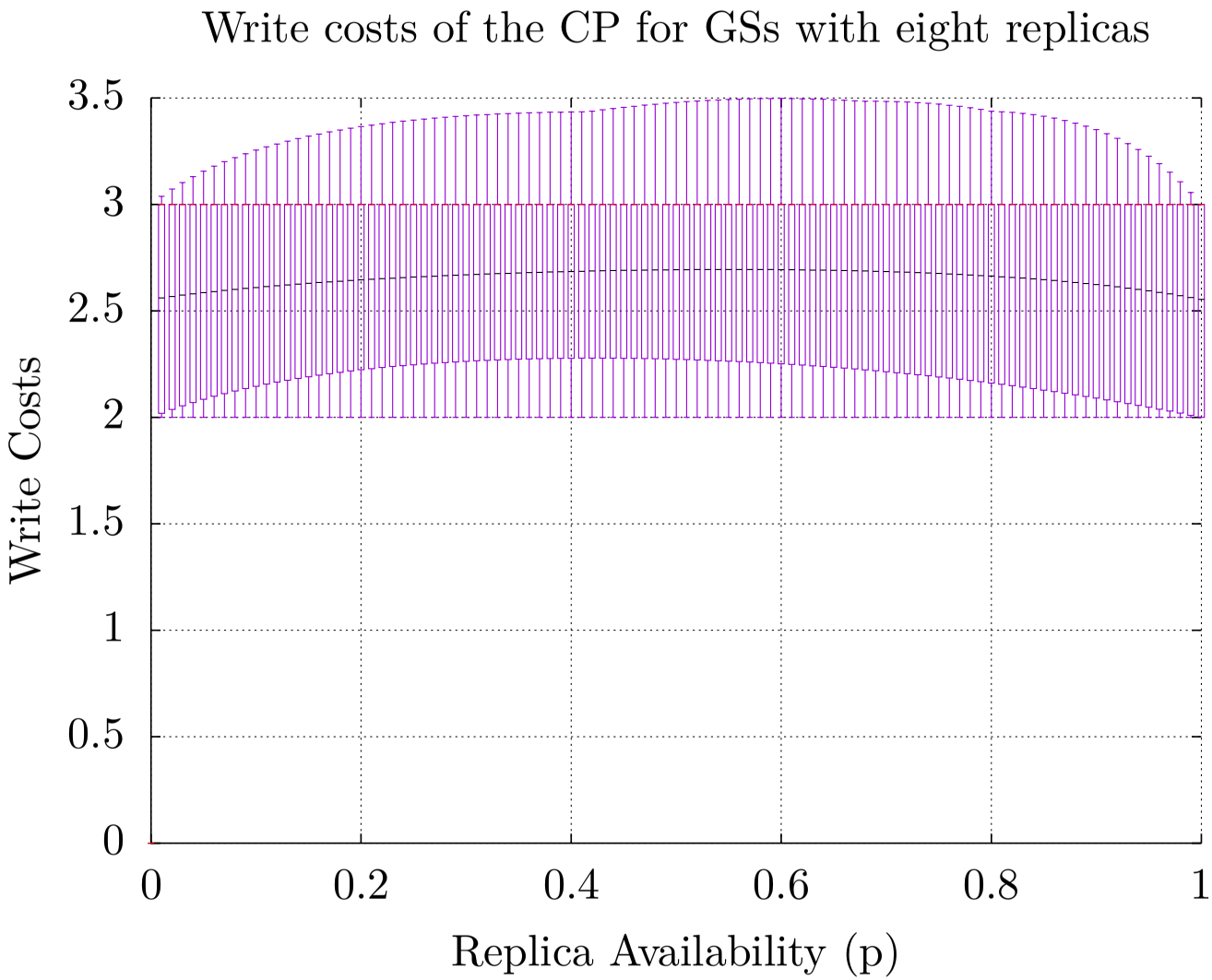


Figure 7.16: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_w(p)$ of the CP with eight replicas. 6169 of the 50000 tested simple, non-isomorphic GSs where usable by the CP.

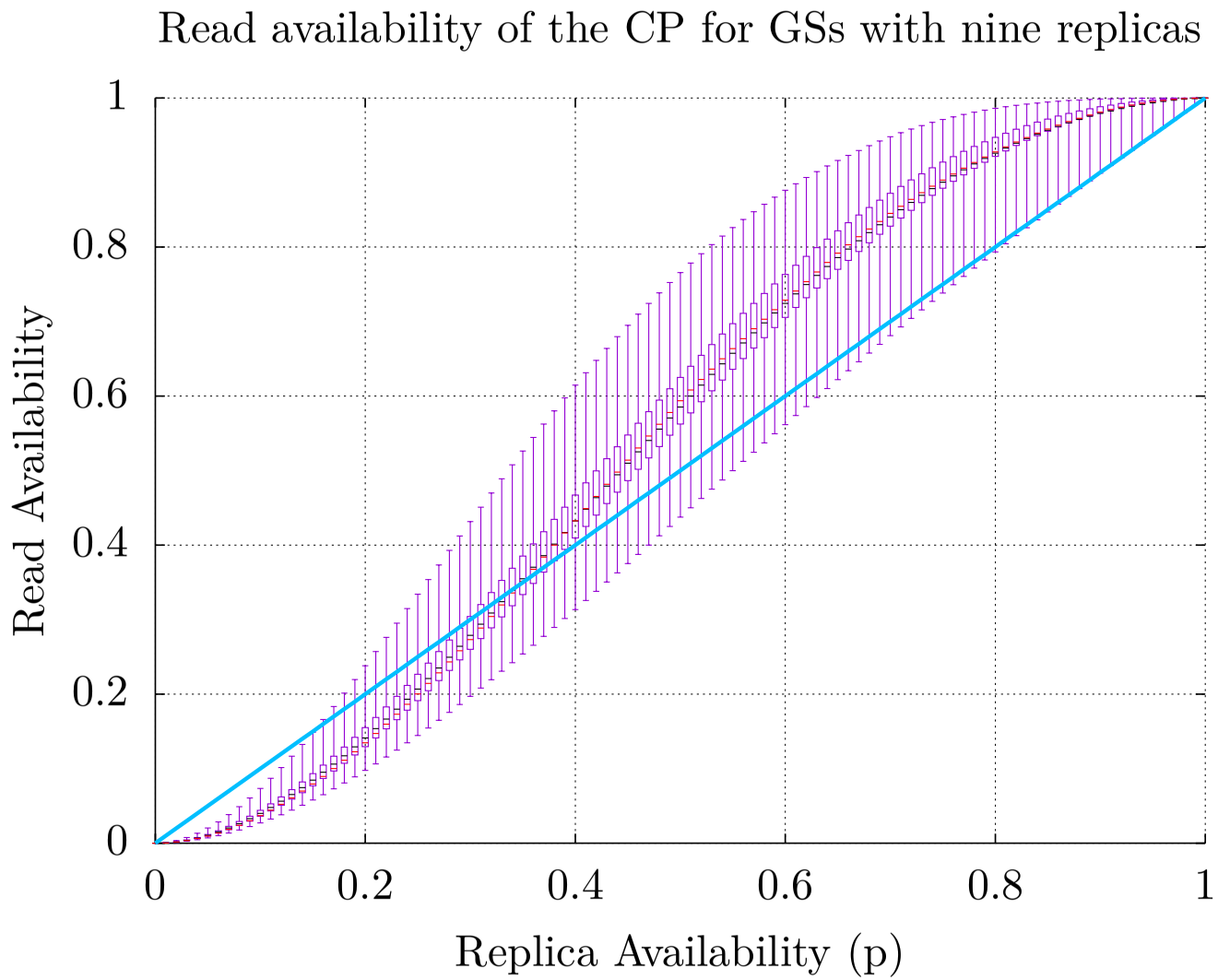


Figure 7.17: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ of the CP with nine replicas. 1555 of the 50000 tested simple, non-isomorphic GSs where usable by the CP.

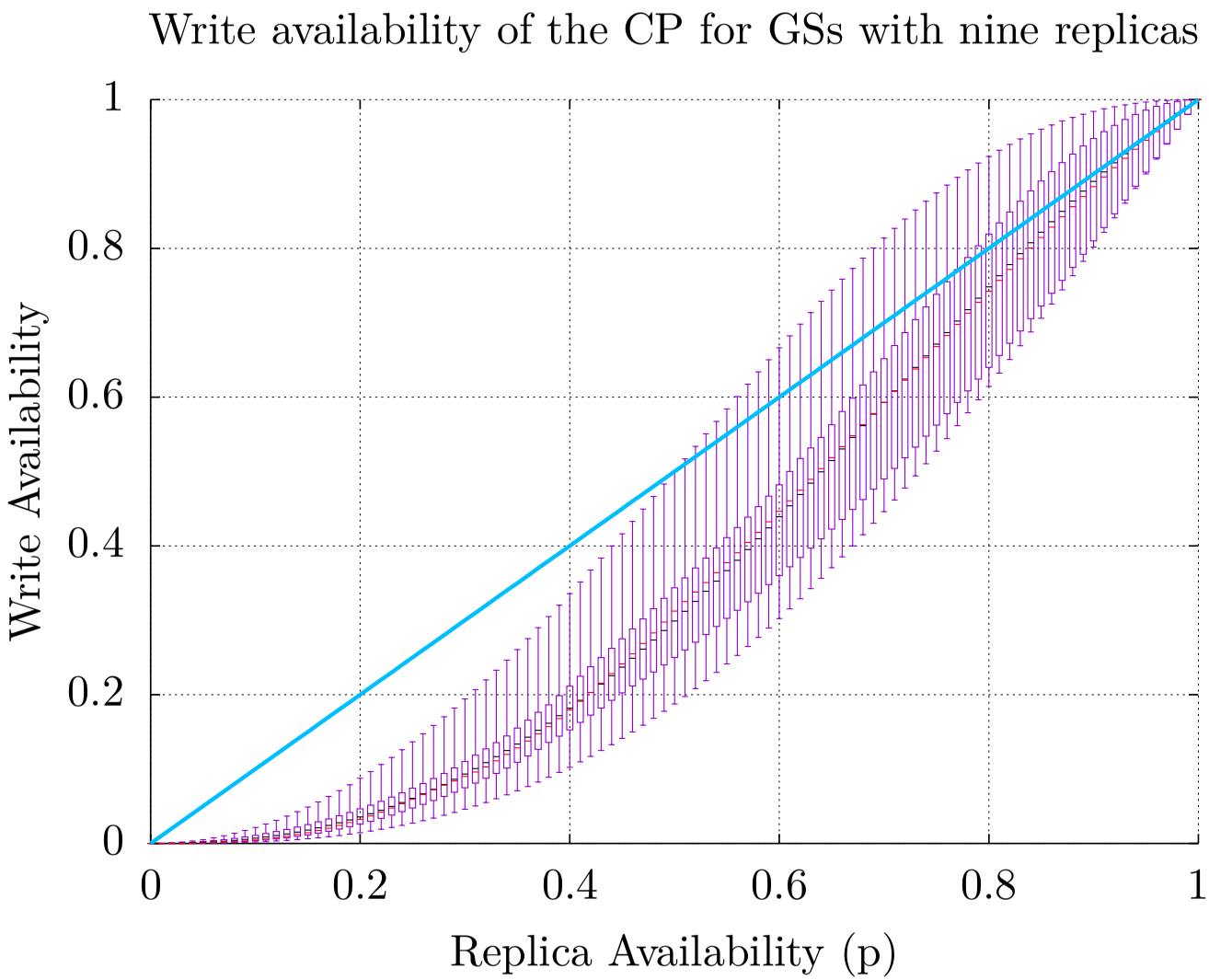


Figure 7.18: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_w(p)$ of the CP with nine replicas. 1555 of the 50000 tested simple, non-isomorphic GSs where usable by the CP.

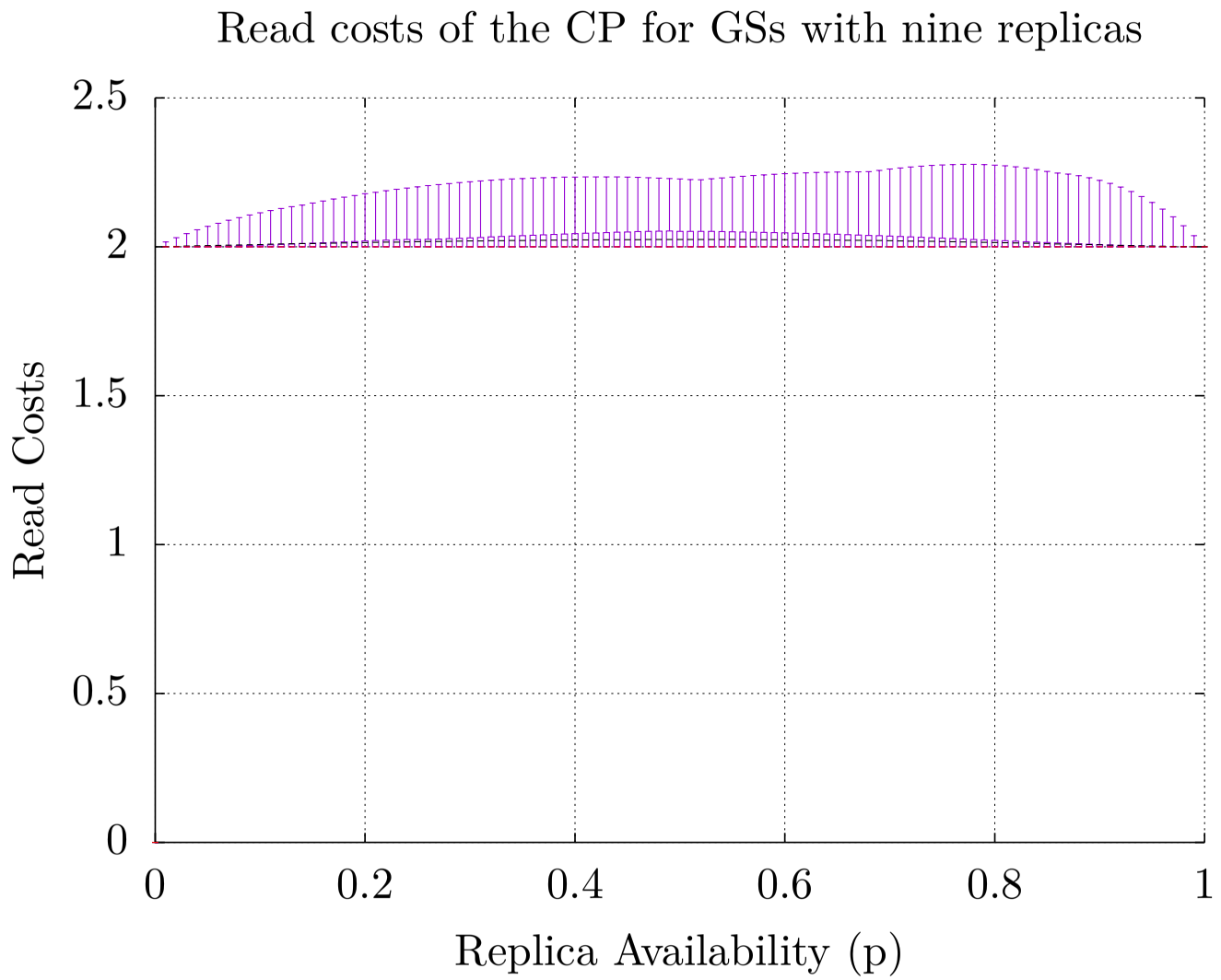


Figure 7.19: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ of the CP with nine replicas. 1555 of the 50000 tested simple, non-isomorphic GSs where usable by the CP.

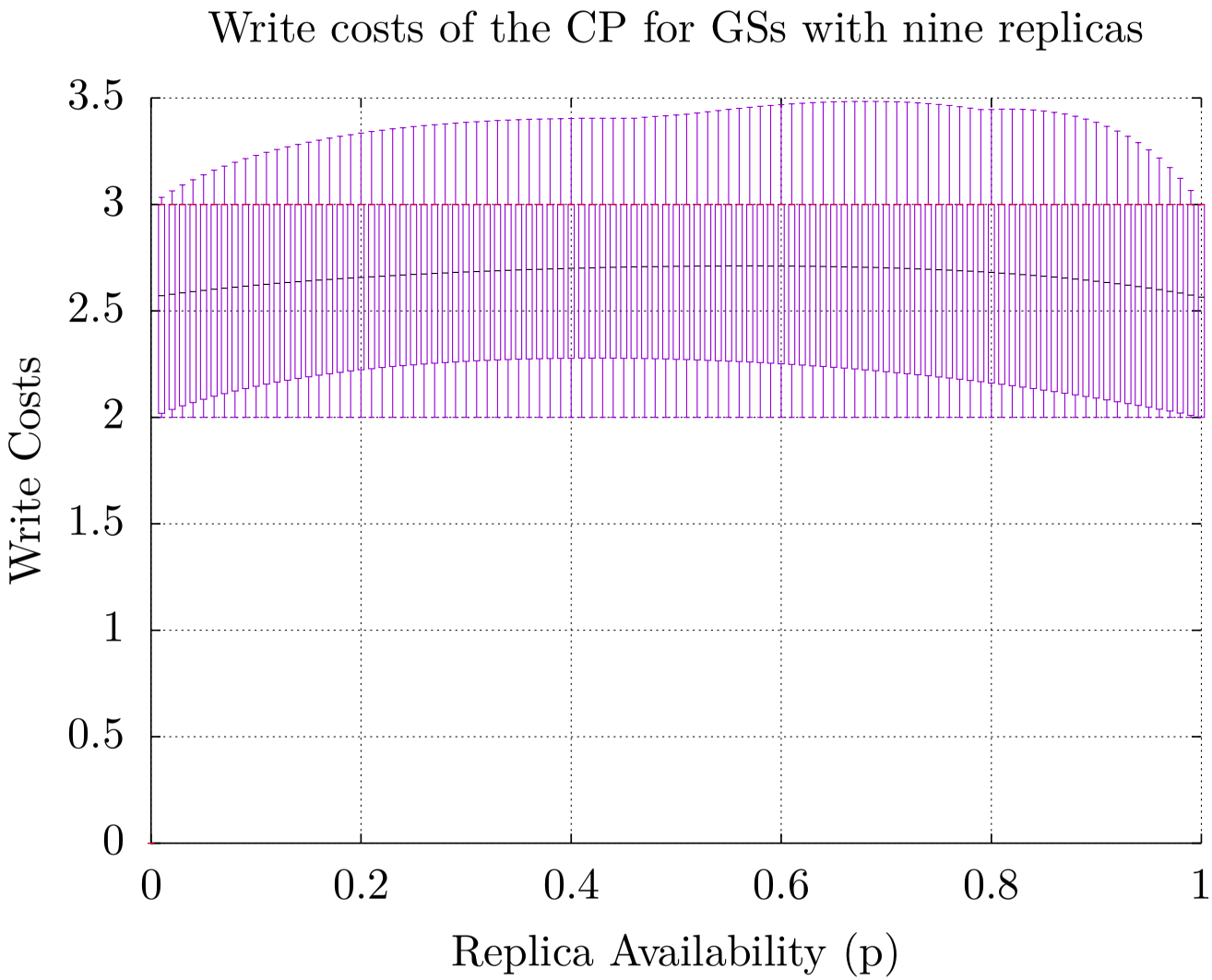


Figure 7.20: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_w(p)$ of the CP with nine replicas. 1555 of the 50000 tested simple, non-isomorphic GSs where usable by the CP.

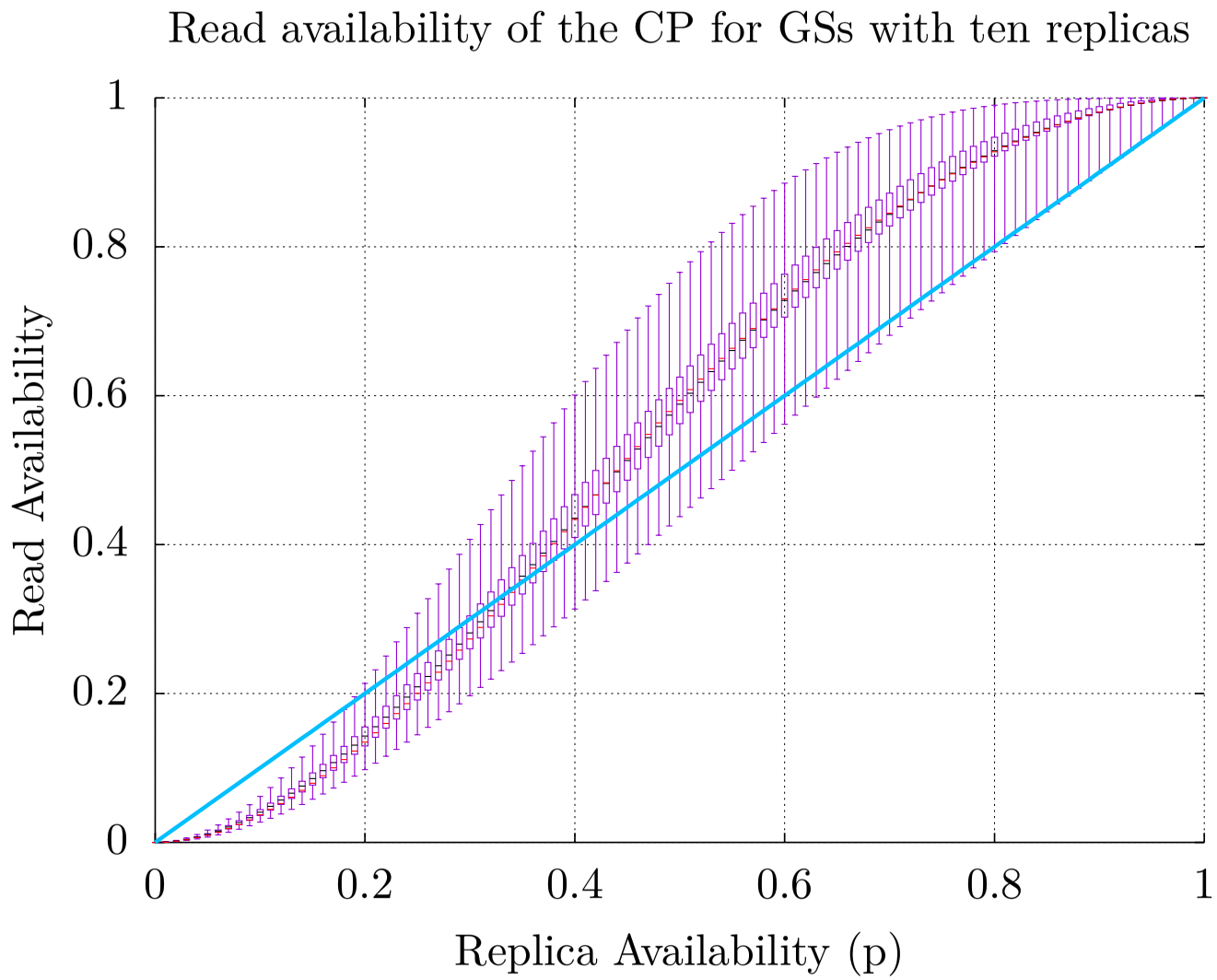


Figure 7.21: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_r(p)$ of the CP with ten replicas. 200 of the 50000 tested simple, non-isomorphic GSs where usable by the CP.

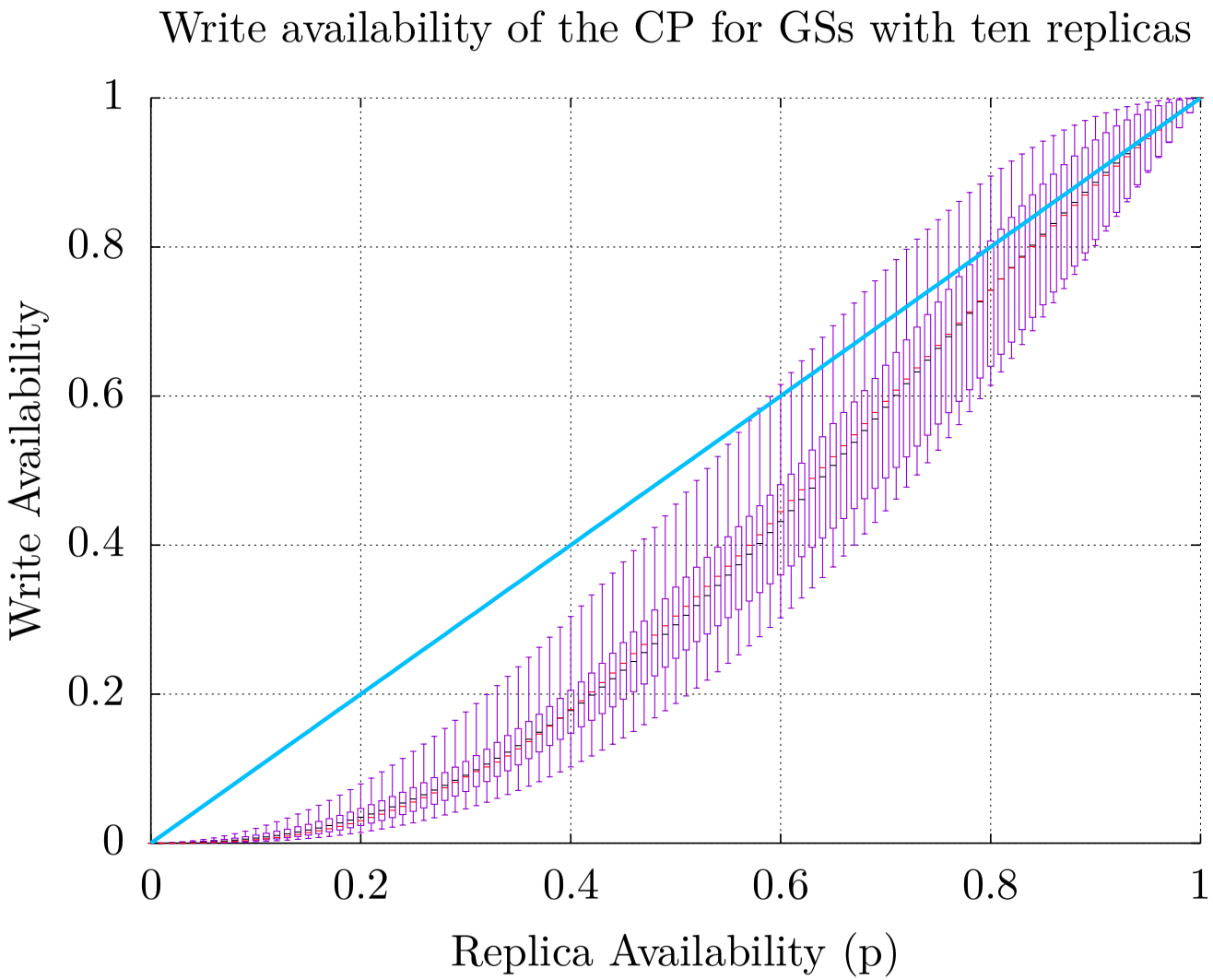


Figure 7.22: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $a_w(p)$ of the CP with ten replicas. 200 of the 50000 tested simple, non-isomorphic GSs where usable by the CP.

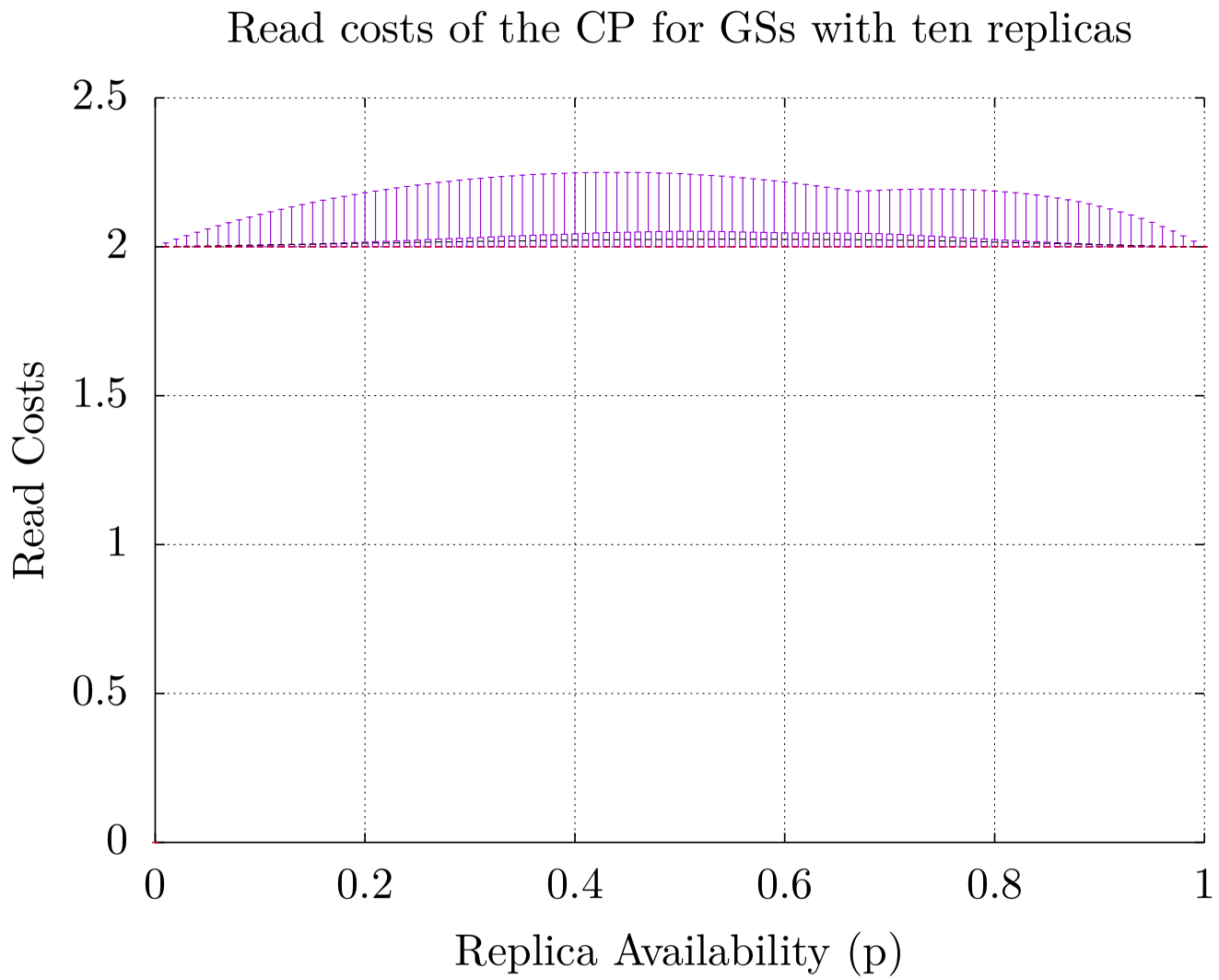


Figure 7.23: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_r(p)$ of the CP with ten replicas. 200 of the 50000 tested simple, non-isomorphic GSs where usable by the CP.

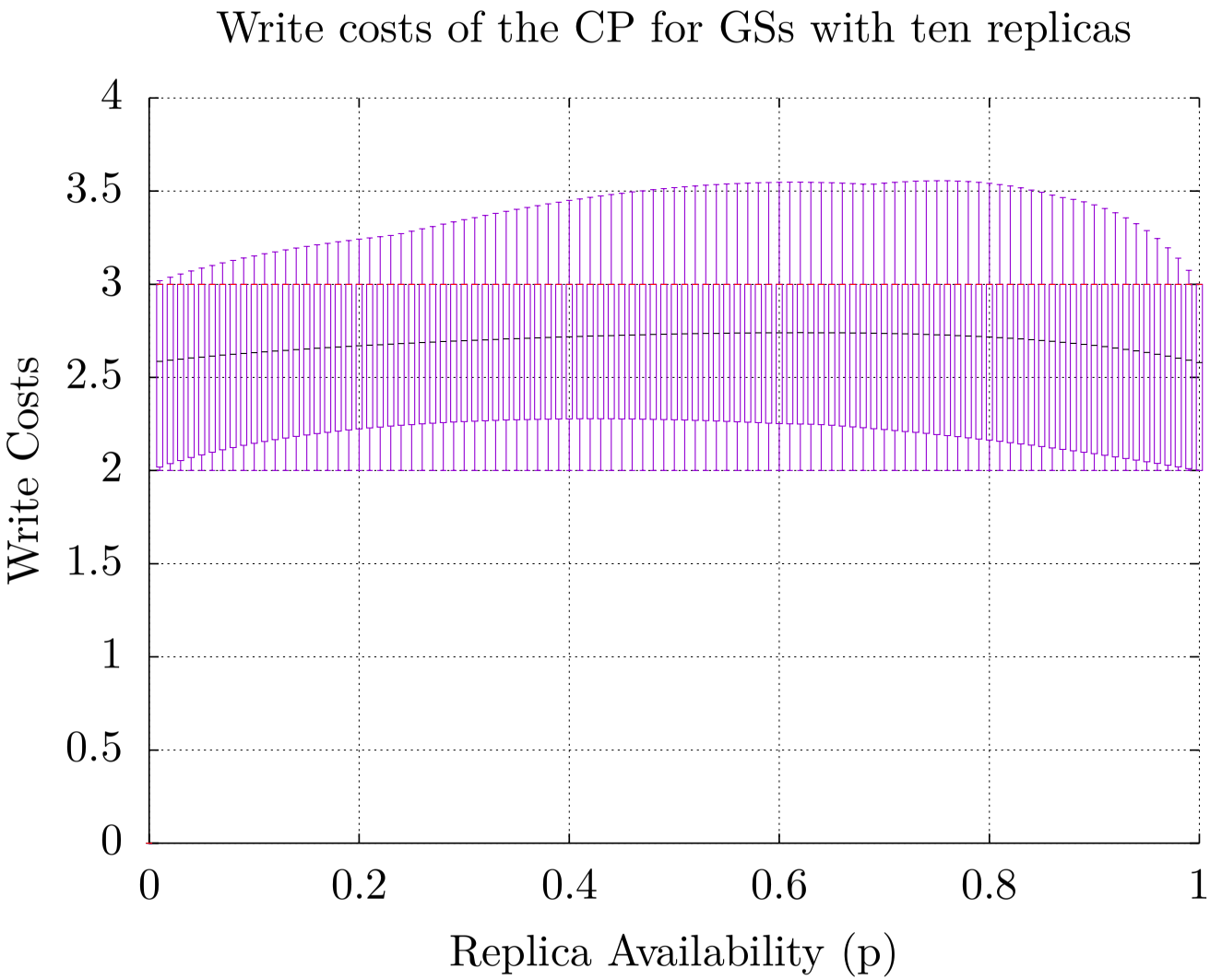


Figure 7.24: The minimal (low whisker), the 0.25 Quantil (box bottom), the median (black line), the average (red line), the 0.75 Quantil (box top), and the maximum (high whisker) $c_w(p)$ of the CP with ten replicas. 200 of the 50000 tested simple, non-isomorphic GSs where usable by the CP.

8 Putting it all Together

So far, multiple ways of bridging the gap between the LNTs and the PNTs have been presented. The mapping approach maps existing QPs to PNTs. The CIP and the CP work directly on a given PNT, but are not always applicable or might not be performing well enough.

The question arises, how to choose which approach to find the best QP for a given PNT.

Algorithm 20 on the following page shows how the best QP can be chosen for a given PNT. The algorithm requires three inputs. The first input is the PNT that should be used as the communication medium. The input *cmp* is a user defined function that yields a value based on the RQS′ and WQS′ of a QP. The value returned by this function is then used to order different QPs when applied to the given PNT. An obvious candidate for such a function is the ARW. The last input is a set of QPs that should be tested as possible candidates for the best QP. Again, what is the best QP is based on the comparison criterion passed into the algorithm. The idea is to applicability of the CIP, the CP, and different QP when mapped on the specified PNT. As mapping are computational expensive, the kNN is first used to determine if finding the best mapping for a given QP is worth it.

On line 1 in Algorithm 20 the best mapping for the MCS is found. This is a not too costly operation, as shown in Subsection 9.4.2 on page 221, as only one of the $N!$ mappings has to be tested for the MCS. The next two lines apply the CIP and the CP to the given PNT. For those three QPs, the *cmp* criterion is used in the comparison. The outcome of these three QPs is then compared with the help of *cmp* as shown on line 4 in Algorithm 20, line 6 in Algorithm 20, and line 8 in Algorithm 20. The next part of the algorithm is only executed if the PNT consists of less than nine vertices. Currently nine vertices is the upper end of what is computational possible with the given mapping implementation. This limit may change in the future. Starting on line 11 in Algorithm 20 all other QP candidates are tested how they compare given the *cmp* criterion. As has seen before, finding the best mapping has a $\mathcal{O}(N!)$ complexity and should be avoided as possible. Therefore, the result of the mapping is predicted on line 12 in Algorithm 20 with the presented kNN approach shown in Section 5.5 on page 110. If the prediction is within one *epsilon* of the currently best QP we continue working with this QP. Where *epsilon* is a user defined value that represents a trade-off between investing computation power and ignoring possible candidates. The algorithm also continues with this protocol if no prediction was possible, for instance, because no historical data was available. Both cases are shown on line 13 in Algorithm 20. This allows to discard QPs, for which the prediction indicate that the they will not perform

Algorithm 20: The process of finding the best QP for a given PNT

```
   Input: pnt = the PNT to find the best QP for
   cmp = the comparison criterion for QPs
   qps = a set of available QPs
   epsilon = cutoff value for knn
   Result: the best QP
 1 mcs = bestMapping(pnt, cmp, MCS)
 2 cip = circleProtocol(pnt, cmp)
 3 cp = crossingProtocol(pnt, cmp)
 4 best = mcs
 5 if cmp(cip) > cmp(cp) ∧ cmp(cip) > cmp(mcs) then
 6 |   best = cip
 7 else if cmp(cp) > cmp(cip) ∧ cmp(cp) > cmp(mcs) then
 8 |   best = cp
 9 end
10 if |pnt| ≤ 9 then
11 |   foreach qp ∈ qps do
12 |   |   knn = predict(qp, pnt, cmp)
13 |   |   if |(cmp(knn) − cmp(best)| < epsilon ∨ knn = ∅ then
14 |   |   |   qpMapped = map(qp, pnt, cmp)
15 |   |   |   storeMappingResult(qpMapped, qp, pnt)
16 |   |   |   if cmp(qpMapped, best) then
17 |   |   |   |   best = qpMapped
18 |   |   |   end
19 |   |   end
20 |   end
21 end
22 return best
```

well on the given PNT. The BC_{max} feature will be used for the predictions. This feature was selected based on its good results as shown in Subsection 5.5.4 on page 121. If a QP is tested further, the best mapping for it is computed and then it is compared with the current best QP. This is done for all the QPs part of *qps*. To build up a store of historical results, the *k*NN predictions can build upon, the mapping results are stored as shown on line 15 in Algorithm 20. Finally, *best* is returned as the best possible QP for the given PNT under the given comparison criterion *cmp*. line 22 on the facing page represents a practical approach for finding the best QP for a given PNT. In an ideal world the algorithm would continue to test all QPs even with PNTs with a growing number of vertices. If and when fast mapping implementation become available this algorithm can be easily adapted to make use of them and thereby expand its applicability.

9 Performance Optimization of the Analysis Program

To analyze the present QPs, and techniques, two options exist. The first possibility is to do the computation by hand. This is possible for QPs with a closed formula but becomes increasingly error prone and extremely time consuming for mappings. The second approach is to write a program to do these analyses. This approach is also error prone and time consuming. But the place of error is moved. Assuming the used formulas are correct, the "by hand" approach usually leads to errors, made by the computer[1] during the computation. If the error is spotted, the computer has to recompute the analyses. This is again error prone and maybe even more important, frustrating. This frustration, then likely leads to more errors.

If there is an error with the implemented program, the error is fixed and the program is run again. The biggest benefit of creating a program to facilitate these analyses, is to use it on different inputs. For instance, in the analyses of the mapping-approach many different QPs are mapped to many different PNTs. It would be silly to create a new program for each analysis. Creating a program that takes the PNT and the GS as input is not that much more complex, but dramatically decreases the number of programs that have to be created.

Sometimes, programs that naively implement the formulas, required in the analyses, still execute too slow. In case of this thesis, the original C++ implementation would have required about seven months to analyze the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ for the TLP with nine replicas mapped to a PNT also with nine replicas. This runtime would have made testing many QPs mapped to many PNTs impossible. Therefore, optimizations to decrease the analysis time had to be developed.

In this chapter the different techniques, guidelines and tools used to increase the performance of the program developed for the analyzes in this work are presented.

9.1 Preliminaries

The original program, called middcir, was written in C++. It was superseded by middcir2, which in turn was written in D. The C++ version was dropped as compilation of this version took multiple times longer than running its integrated tests and benchmarks. The compile-time (CT) of the D version is negligible. An Intel x86-64 central processing units (CPUs) are used to execute middcir2 (MC).

[1] Computer in this case refers to the person doing the computation

Therefore, D and asm in the Intel-syntax for listings and demonstration purposes are used in this chapter. It is refrained from giving a general introduction into both languages, as this is out of the scope of this work, and only a limited subset of both languages is required which will be explain on first appearance.

The performance of a program depends on many parameters. Some of the parameters are known, some are kept secret by the CPU vendors, and others have been partially re-engineered. On top of that, those parameters interact with each other. As some information are just not obtainable, no precise statement can be made of how the fastest program for a given purpose looks like.

Therefore, the goal is to create a program that is fast enough to compute the wanted result in the available time. Figure 9.1 shows the optimization approach

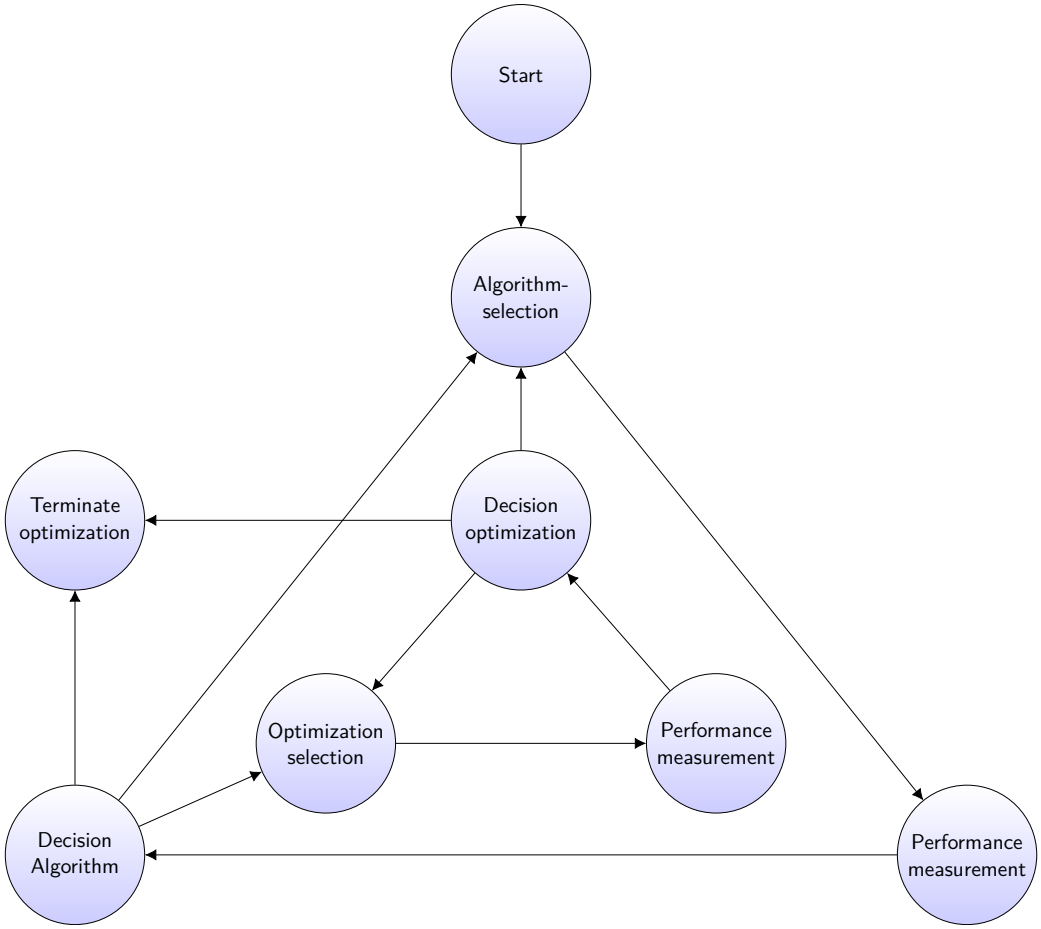


Figure 9.1: Optimization approach used for this work.

used in this work. For each computational task, this approach is followed. Initially, an algorithm is chosen that solves the given problem. Then, the performance of that algorithm is measured. The step labeled Decision Algorithm follows. There are three possible decision options in this step. The first is to terminate the

optimization. This path is taken if the current implementation of the chosen algorithm is fast enough for the particular use case. The second option is to choose another algorithm. This path is usually taken when the performance measurement shows that the performance of the chosen algorithm is nowhere near the required performance. The performance measurement step is a very important step and must not be overlooked. Due to the complexity of the process, the rule of thumb is that performance predictions, made by humans, are almost always wrong. The only way to spot performance problems is therefore to measure the performance. A common method of measuring is taking the time a function takes to finish a unit of work. As measuring accurately is more difficult than it might appear, Subsection 9.1.1 explains the used method of measuring in more detail. The last option to take, is going to the node Optimization selection. This choice starts a new cycle, in which lower level optimization techniques are applied to the previously chosen algorithm. Again, this cycle can be iterated many times over. The chosen performance optimizations highly depend on the chosen algorithm and its data structures. Not all optimizations, which are presented later, can always be applied. Additionally, their impact on the performance may vary and sometimes its impact also may be negative [32]. This is where Subsection 9.1.1 becomes important. The only way to know if an optimization was successful is to benchmark the performance of the program with and without it.

9.1.1 Benchmarking

In Figure 9.1 the task Performance measurement appears twice. Performance measuring is an important part of the optimization process. Without good data, it is hard to make the correct decision regarding the optimization process.

All benchmarking has to be done with programs that where optimized by the compiler.

Likely, the first used tool is the unix `time` command. It returns how long it took a program to execute from start to finish. Usually, this execution time does not only depended on the performance of the program, but also on factors like how often the operating system schedules the program to be executed. To mitigate this scheduling influence, the program can be executed multiple times and the individual execution times aggregated and averaged. This might be unfeasible if the program has a very long execution time.

Instead of executing the complete program, it might be faster to measure the performance of functions individually. Using the `time` command for this purpose becomes tedious, fast. It is easier to measure the execution time from inside the program. The first question to answer is which function should be benchmarked. This can be answered by tools like valgrind [33] or perf [34]. These tools allow to monitor and visualize how long and how often each function of a program is executed. Figure 9.2 on the next page shows a part of a call-graph of a benchmarked program. Each box represents a function. Not all called functions are visualized as they may have been inlined into other func-

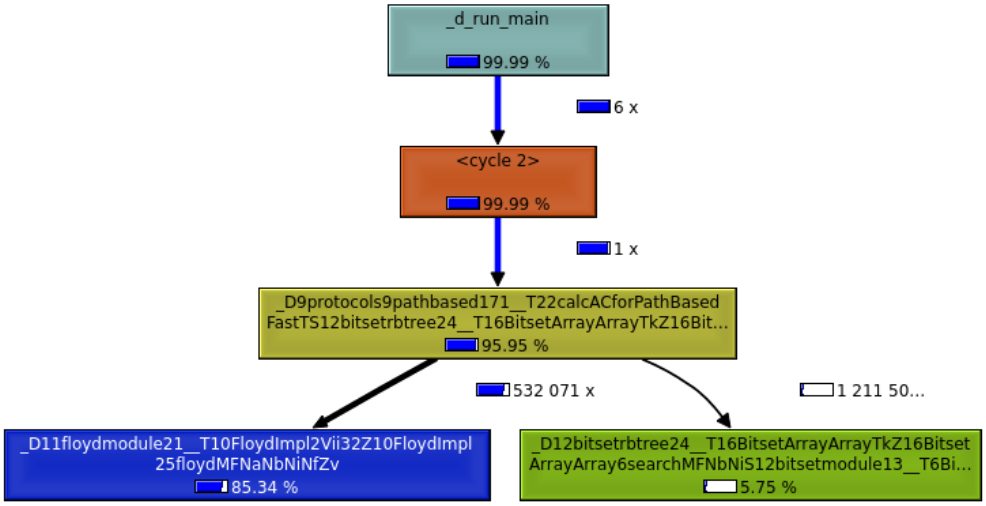


Figure 9.2: Part of the visualization of the call graph of an execution of a benchmarked program.

tions calling them. The arrows show, which functions call which other function. The tool only shows the names of the functions in their mangled form, for instance `_D9protocols9pathbased171_T22calcACforPathBasedFastTS12bitsetrbtree24_`. A demanagle tool reveals the function name `calcACforPathBasedFast`. The figure shows that 85% of the execution time is spent in this function and that it is called 532071 times. The number of calls to it in combination with the time spent in it make it a good candidate to investigate possible performance improvements.

This is sometimes called the hot spot analysis [35]. The identified function is therefore called a hot spot.

For the sake of example, let the function shown in Listing 9.1 to be an identified hot spot.

```
1 int fun(int a, uint b) {
2     int ret = 0;
3     for(uint i = 0; i < b; ++i) {
4         ret += a;
5     }
6     return ret;
7 }
```

Listing 9.1: A function identified to be a hot spot.

Function `test1` shown in Listing 9.2 shows how the function `fun` is benchmarked.

```
1 void test1() {
2     enum numRuns = 3001;
3     auto rnd = Random(numRuns);
4     Duration[numRuns] durs;
5     StopWatch sw;
```

```
6
7       for(int i = 0; i < numRuns; ++i) {
8           sw.reset();
9           int a = uniform!int(rnd);
10          uint b = uniform(0U, 10000U, rnd);
11
12          sw.start();
13          int t = fun(a, b);
14          sw.stop();
15
16          durs[i] = sw.peek();
17      }
18
19      sort(durs[]);
20
21      writefln("min: %15d | avg: %15.4f | median: %15d | max: %15d",
22          durs.front.total!"hnsecs"(),
23          sum(durs[], Duration.init).total!"hnsecs"() / cast(double)
                numRuns,
24          durs[numRuns / 2].total!"hnsecs"(),
25          durs.back.total!"hnsecs"());
26  }
```

Listing 9.2: The function `test1` is a function which is benchmarking the function `fun` shown in Listing 9.1.

In line 2 in Listing 9.2 the number of iterations is set to 3001. This means that function `fun` will be run 3001 times. The number 3001 is chosen for two reasons. The first is to run the benchmark often enough that meaningful impression of the performance of the function are obtained. The second reason is not to spend too much time benchmarking a single function. There are usually many functions that make up a program and many of them might require optimization to make the program as a whole performant. So, the time spend optimizing has to be distributed among all hot spots. An uneven number is choose to simplify the median computation, in this example, on line 24 in Listing 9.2. The type `StopWatch` allows to measure the time between a call to `start` and `stop` as shown in line 12 in Listing 9.2 and line 14 in Listing 9.2. Starting on line 9 in Listing 9.2 the function `uniform` is used to generate a random `int` and a random `uint`. These values depend on the state of the random number generator created on line 3 in Listing 9.2. The random number generated needs a specified seed value, such that it can reproducibly generate the same test data. If the test data is not the same between different runs of the benchmark, comparing benchmark results would be pointless, as it cannot be known if the performance optimizations were successful or the test data was simply favorable. After the function `fun` has returned, the method `peek` of the `StopWatch` is used type to record the duration `fun` took to execute, as shown on line 16 in Listing 9.2. Eventually, the minimum, the average, the median, and the maximum execution time are printed. The times are printed as hecto-nanoseconds. From experience, hecto-nanoseconds are the maximum resolution that is reliably measurable with modern Intel CPUs. When looking at

benchmarks and the benchmark functions, it is seen that function result is not used nor does the benchmarked function produce any side effect. This leads to a severe problem. When this benchmark is compiled, the compiler might consider the call to the function `fun` as dead code and, in turn, removes it. This only happens when the compiler is told to optimize the program. Consequently, the benchmark has to make sure that the compiler cannot remove the call to `fun`. The easiest way to achieve this is to print the result of the function call. But printing to an output device is slow and quickly becomes unhandy when multiple benchmarks are compiled into the same program.

```
1 void doNotOptimizeAway(T)(auto ref T t) {
2     if(thisProcessID() == 0) {
3         writefln("%X", cast(ulong)(cast(void*)(&t)));
4     }
5 }
```

Listing 9.3: The function `doNotOptimizeAway` makes sure that if the passed parameter `t` is a return value of a function `a`, then the compiler cannot remove a call to function `a` that produced the passed value.

To trick the compiler into thinking that the computed value is printed, and requiring its computation, the function `doNotOptimizeAway` shown in Listing 9.3 is used. The function `doNotOptimizeAway` will print the address of the passed value if the process ID of the benchmark program is zero. At least on Linux, BSD, and Windows this process ID is not used for user processes. As long as the binary resulting for the benchmark is run as a user process, the address of the value will never be printed. This information about the return value of the function `thisProcessID` is unknown to the compiler, therefore it cannot remove the call to the function that produces the value and in turn, the call to the benchmarked function can not be removed either.

```
1 void test2() {
2     enum numRuns = 3001;
3     auto rnd = Random(numRuns);
4     Duration[numRuns] durs;
5
6     StopWatch sw;
7
8     for(int i = 0; i < numRuns; ++i) {
9         sw.reset();
10        int a = uniform!int(rnd);
11        uint b = uniform(0U, 10000U, rnd);
12
13        sw.start();
14        int t = fun(a, b);
15        sw.stop();
16
17        durs[i] = sw.peek();
18
19        doNotOptimizeAway(t);
20    }
```

```
21
22      sort(durs[]);
23
24      writefln("min: %15d | avg: %15.4f | median: %15d | max: %15d",
25          durs.front.total!"hnsecs"(),
26          sum(durs[], Duration.init).total!"hnsecs"() / cast(double)
                numRuns,
27          durs[numRuns / 2].total!"hnsecs"(),
28          durs.back.total!"hnsecs"());
29  }
```

Listing 9.4: The function `test2` is a function which is benchmarking the function `fun` shown in Listing 9.1 and that uses the function `doNotOptimizeAway` to force the compiler to not remove the function call to `fun`.

9.4 finally shows the exemplary use of the benchmarking code.

9.2 Modern CPU Architectures

Modern CPU architectures (> 2006) have many features that increase the performance of a given program. Most of these features are not directly accessible through the programming language used. Those features are only indirectly accessible by writing programs in such a way that the compiler used can generate assembly code that allows the CPU to execute that assembly code in a way that facilitates the CPU features in an optimal way. Figure 9.3 on the next page shows a simplified version of the Intel Broadwell architecture. It shows only a single core of the multicore architecture. The blue, dashed rectangle in Figure 9.3 shows the boundary of a core. The red, dotted rectangle encloses the ports, also known as the execution units of a core. To better understand the architecture in Figure 9.3 some terms need to introduced. The L1 instruction cache (L1IC) is a so-called level 1 instruction cache. Lower cache level are faster to access by a port. The L1IC has 32KiB of memory that is organized in lines of 64 Bytes. Therefore, the L1IC is organized into 512 lines. CPU caches usually are not able to store individual bytes, but rather store x consecutive bytes. The L1IC is an 8-way-associative cache. This means that each 64Byte block of memory, identified by a unique address, can be stored in one of eight different lines of the cache. In a completely associative cache each block of memory could be stored in any cache line, this would decrease contention but makes the cache lookup slower. Such a 8-way-associative cache is therefore a trade-off between lookup speed and contention avoidance. The L1 data cache (L1DC) is similar to the L1IC, except that it caches data and not instructions. Both the L1IC as well as the L1DC get their data from the L2 cache. In the broadwell architecture, this is a 256KiB big cache. Both caches are able to load one cache-line per cycle. The L2 cache is fed by the L3 cache. This cache is shared between each core of the CPU. The L2 cache can move 32 Bytes per cycle from and to the L3 cache. The L3 cache stores and loads its data from memory. Both operations have a latency of 46 up to 65 cycles,

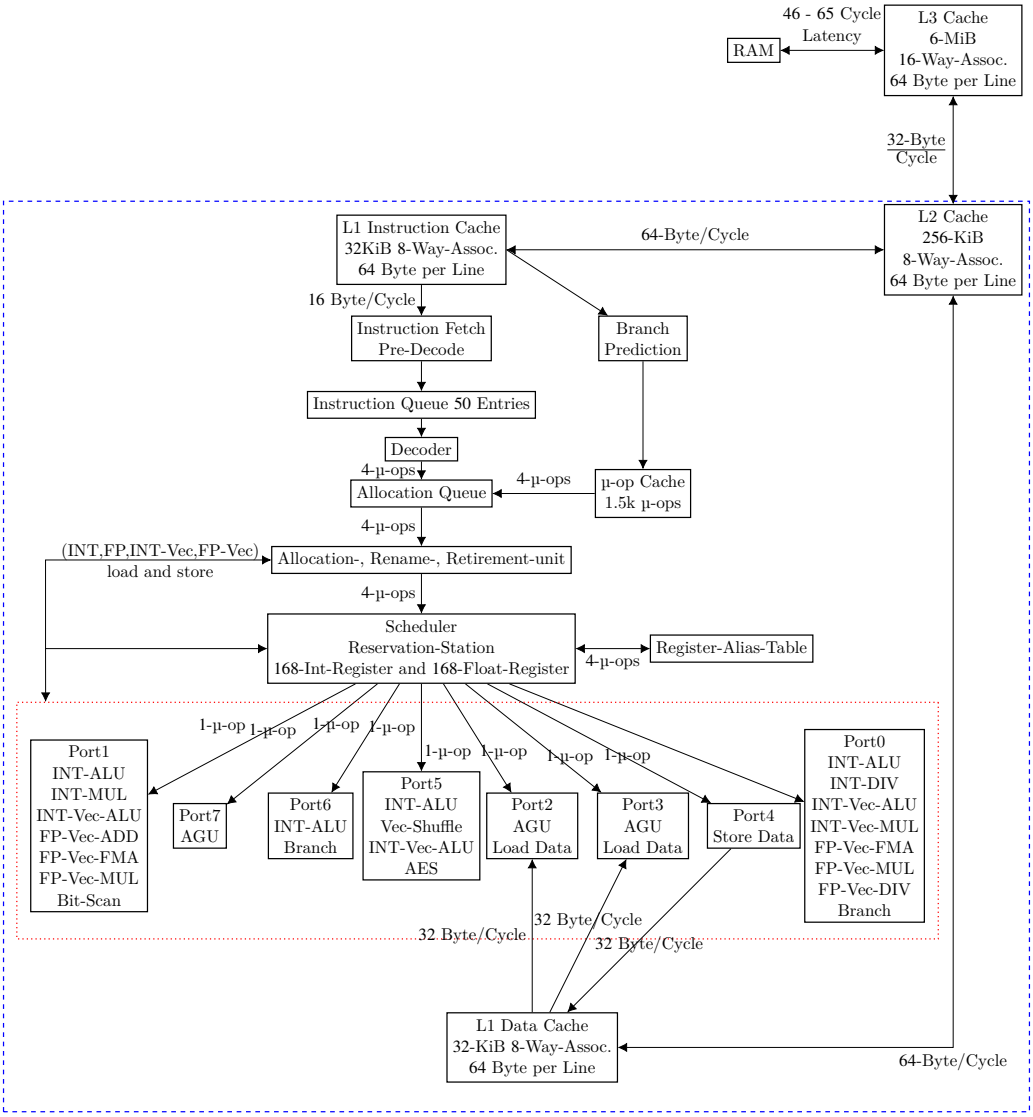


Figure 9.3: A simplified Intel Broadwell architecture overview [36, 37, 38].

depending on the platform, and are able to achieve a bandwidth of about 76GiB/s [39].

The red, dotted box encloses the ports. Each has one or more possible types of instructions it can execute. The ports are executing their operations in parallel to each other. Which instruction is to be executed by which port is scheduled by the Scheduler. The operations in Figure 9.3 are abbreviated. The following descriptions give an explanation of these abbreviations.

INT-ALU Integer arithmetic logical unit. These are instructions like plus, minus, cmp, etc. working on integer types.

INT-MUL Integer multiplication. This instruction multiplies integer values.

INT-DIV Integer division. This instruction executes integer division.

INT-VEC-ALU Integer vector arithmetic logical unit. The single instruction multiple data (SIMD) variant of the INT-ALU instruction.

FP-Vec-Add Floating point vector addition. This set of instruction adds floating point vectors.

FP-Vec-FMA Floating point vector fused multiply and add. This set of instruction adds and multiplies floating point vectors.

FP-Vec-MUL Floating point vector multiplication. This set of instruction multiplies floating point vectors.

FP-Vec-DIV Floating point vector division. This set of instruction executing division on floating point vectors.

Vec-Shuffle Vector shuffle. These instructions allow to change the position of the individual members of integer vectors as well as floating point vectors.

Bit-Scan These instructions are able to scan bits for certain properties.

AGU Address generation unit. The Intel x86 based architectures are capable of complex address calculations. These are done by the AGU.

Branch These instructions are conditional jump instruction.

Load Data These are instruction that load data from memory addresses.

Store Data These are instruction that store data from memory addresses.

All ports share 168 integer and 168 floating point registers. The integer registers store 8 bytes and the floating point register store 32 bytes each. The floating point registers are also used by the INT-VEC instructions. The Instruction Fetch, Pre-Decoder, Instruction Quere, Decoder, and Allocation Queue translate assembly instruction into so-called µ-operations.

The Branch Prediction tries to predict µ-operations that will be executed in future cycles. This helps to keep all ports and other resources busy. The premise here is, that un-utilized ports are to be avoided, even when the outcome of speculatively executed operations might not be needed after all. This speculative execution of instruction was exploited in the infamous spectre and meltdown security vulnerabilities [40, 41].

Therefore, the low-level optimization target is to keep all data as well as all code in their respective L1 caches. This allows all ports to have code and data to execute. This is done by avoiding branching operations and by easily predictable memory accesses. The most predictable memory access is the iteration of an array from front to back.

Now that the used CPU-Architecture is known, the different stages of the compilation and execution can be inspected as well how influence on these stages can alter the performance of the program.

9.3 Places for Contact

Figure 9.4 shows a simplified process that explains how source code is eventually executed on a CPU. Some of the nodes in that figure are labeled with letters. For each of these letters, possible optimizations are presented. Starting with e), the

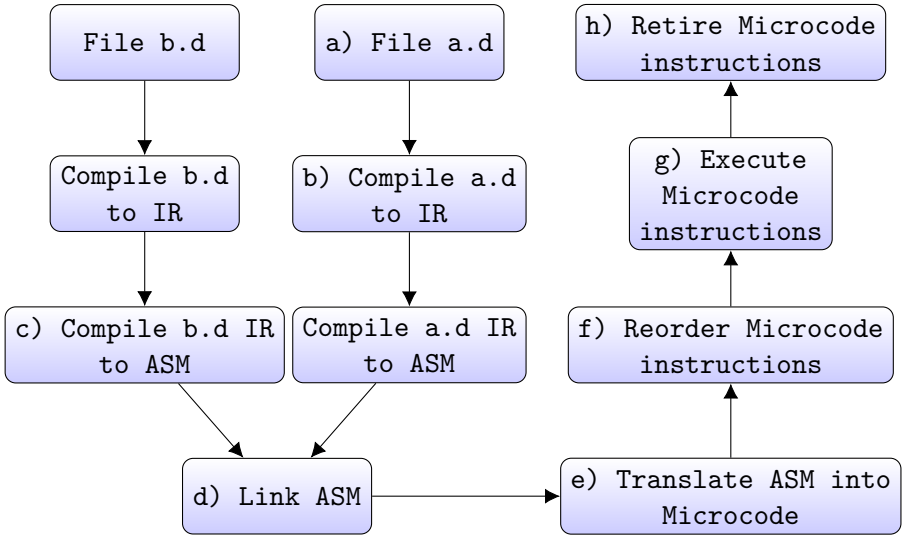


Figure 9.4: Simplified process that describes the steps from a source file to execution on an CPU.

CPU starts executing the compiled program.

9.3.1 a) Precompilation Optimizations

A simple approach to increase runtime performance is to move computation to CT. Listing 9.5 shows an example of this approach. The function `gcd` computes the greatest common divisor.

```
1  int gcd(int a, int b) {
2      int temp;
3      while(b != 0) {
4          temp = a % b;
5          a = b;
6          b = temp;
7      }
8      return a;
9  }
10
11 unittest {
12     int gcd1 = gcd(32, 8);
13     enum gcd2 = gcd(32, 8);
14
15     assert(gcd1 == 8);
16     assert(gcd2 == 8);
17 }
```

Listing 9.5: Example of moving computation to CT.

On line 12 in Listing 9.5 the function is executed at runtime. On line 13 in Listing 9.5 the function is executed at CT. In D the keyword enum evaluates an expression at CT if possible [42].

As this is an idealized example, such an optimization is not always possible to such an extend, but in middcir2 (middcir2) there are examples where this idea is used. During a profiling session, it was found that about 50% of the runtime was spent on the power function. The power function was called extremely often during the evaluation of Equation 3.13. Looking closer revealed that the expression

$$p^{|q|}(1-p)^{N-|q|} \tag{9.1}$$

was the source of these frequent calls. Ideally, the computation of this expression can be avoided at runtime or at least move parts of it to CT. This was be done by creating a lookup-table for this expression. This expression has three inputs p, q, and N. Mathematically, $0.0 \leq p \leq 1.0$, but in order to plot the results $p \in \{x \in \mathbb{N}_0 | 0 \leq x \leq 100 \wedge \frac{x}{100}\}$ are needed. This set shows that 101 p values are evaluated. N is the number of replicas used by the QP. From practical experience, it is known that QP with more than 32 replicas will not be analyzed, therefore $1 \leq N \leq 32$. With N given $|q| \leq N$.

```
1 enum ps = 101;
2 enum N = 32;
3 enum q = 32;
4
```

```
5 double[ps][q][N] p_lookup = { ... };
```

Listing 9.6: Availability lookup-table

Listing 9.6 on the preceding page shows the structure of the resulting lookup-table. The array with the name `p_lookup` is the resulting lookup-table[2]. The array declaration is read from right to left, meaning the elements of `double[ps]` are aligned next to each other in memory. When using this array, it is made sure that the sub-arrays are iterated from right to left. This way, the chance for cache-hits is improved.

The result of this optimization was that the calculation of the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ required less than 1% of the total runtime. Originally, it was above 50%.

9.3.2 b) Compile to intermediate representation (IR)

Most modern compilers transform the source code of a program into an intermediate representation (IR) [43, 44]. In the case of the used D compiler LDC, which is using the LLVM backend, this IR is a static single assignment (SSA) [45]. If the compiler is set to optimize, the compiler in-build optimizations are applied to the IR [46]. Similar to branch prediction, which will be explained in Subsection 9.3.6 on page 215, it is helpful for the compiler to have programs with as few branch statements as possible. This is because for many of the optimizations, the compiler uses a control flow graph (CFG). The size of the CFG the compiler inspects for optimizations, is limited. The compiler therefore only inspects a moving window of the CFG while optimizing. Having less branch statements decreases the size of the CFG, and thereby allowing the compiler to view a larger part of the complete program leading to potentially better optimizations. Influencing the compiler's in-build optimizations is an extremely complex topic only touch briefly in this work.

Function inlining (FI) describes a process where the body of a called function is basically duplicated at every place of call. This has the advantage that the code of the inlined function is right next to the caller function in the binary. This increases the chance of a cache hit for the instructions of the inlined function. Listing 9.7 and Listing 9.8 on the next page show the function `gcdXGcd` before and after inlining the function `gcd`.

```
1 int gcd(int a, int b) {
2     int temp;
3     while(b != 0) {
4         temp = a % b;
5         a = b;
6         b = temp;
7     }
8     return a;
```

[2]The three dots represent the values calculated at CT.

```
}

int gcdXGcd(int a, int b) {
    return gcd(a,b) * gcd(a,b);
}
```

Listing 9.7: Before function inlining (FI)

```
int gcdXGcd(int a, int b) {
    int ret;
    int temp;
    int a_copy = a;
    int b_copy = b;
    while(b_copy != 0) {
        temp = a_copy % b_copy;
        a_copy = b_copy;
        b_copy = temp;
    }

    ret = a_copy;

    a_copy = a;
    b_copy = b;
    while(b_copy != 0) {
        temp = a_copy % b_copy;
        a_copy = b_copy;
        b_copy = temp;
    }

    ret *= a_copy;
    return ret;
}
```

Listing 9.8: After function inlining (FI)

This in itself seems like a deoptimization. There is more code, which can be identified as bad. There are more branch statements, and more variables. But there are a few crucial differences. The two function calls to `gcd` were removed. Therefore, the `int`s do not have to be pushed on the stack, resulting in less memory usage. The stack currently stored in the L1 cache may or may not have enough free space to store these `int`s, but this cannot be known in the general case. FI allows to avoid this risk. A bigger problem is that the assembly of the `gcd` function may currently not be stored in any of the caches. Loading the machine code of this function from RAM is likely going to take longer than executing the function itself.

FI is used based on some heuristic. Sometimes these heuristics do not yield the intended result. To force FI, D allows functions to be attributed with `pragma(inline, true)`. This forces the inlining of the following function. Sometimes, when the inlining of a function is counterproductive for the performance, the attribute `pragma(inline, false)` disables the FI for the following function.

9.3.3 c) Compile IR to ASM

In this stage, the IR is transformed into a platform specific assembly. In the case of MC, the target platform is Intel x86-64 with the advanced vector extensions (AVX2) instruction set. Specifying the exact target platform is relevant to the performance as the compiler may choose different instruction depending on the platform. It is also possible to compile against x86 as the target platform. The resulting executable will be able to execute on a x86-64 (AVX2) platform, but it will only use eight of the 16 available registers and only use 32 of their 64-bits registers. Also, AVX2 provides 16 additional 256-bit registers. AVX2 instructions belong to the family of SIMD instructions. These SIMD instructions allow to execute the same instruction on multiple items of data, and in the same time, it would require to execute the instruction on a singular data item.

Listing 9.9 shows a simple function. Listing 9.10 and Listing 9.11 on the facing page show the assemble of that function once with a SIMD instructions and once without.

```
1 void multiply(double[] a, double[] b) {
2     for(size_t i = 0; i < a.length; ++i) {
3         a[i] += b[i];
4     }
5 }
```

Listing 9.9: Example function for explaining SIMD instructions

Both functions are truncated to the important parts. The asm in Listing 9.10 uses the `addsd` and `movsd` operations to add one element of the array `b` to the corresponding element in the array `a`. The compiler already unrolled the loop to do four of these steps add a time. Unrolling describes the transformation of a loop into the series of statements that would have been executed by the loop. This will help in the branch prediction as explained in Subsection 9.3.6. Listing 9.11 is different: it uses the `movupd` operation to load four doubles into the `xmm0` register. `movupd` translate to move unaligned packed double.

```
1  .L1:
2      movsd   xmm0, qword ptr [rcx - 24]
3      addsd   xmm0, qword ptr [rax - 24]
4      movsd   qword ptr [rax - 24], xmm0
5      movsd   xmm0, qword ptr [rcx - 16]
6      addsd   xmm0, qword ptr [rax - 16]
7      movsd   qword ptr [rax - 16], xmm0
8      movsd   xmm0, qword ptr [rcx - 8]
9      addsd   xmm0, qword ptr [rax - 8]
10     movsd   qword ptr [rax - 8], xmm0
11     movsd   xmm0, qword ptr [rcx]
12     addsd   xmm0, qword ptr [rax]
13     movsd   qword ptr [rax], xmm0
14     add     rcx, 32
15     add     rax, 32
16     add     rdi, -4
```

```
17      jne     .LBB0_17
```

Listing 9.10: Assembler for the `multiply` function from Listing 9.9 without SIMD instructions.

This means as much as: "move four doubles from an memory address a that must not be $a \bmod 16 = 0$ to a given register". The `addpd` is then used to add the four doubles behind the address `[r10+rcx]` to the four doubles in `xmm0`. `movaps` is then finally used to move the result back into memory. Here, the a in `movaps` stands for aligned. This works as the asm that was truncated from above label `.L1` made sure that the address stored in `r10` is 16 byte aligned. The offset stored in `rcx` is also always incremented by 16, making each successive address aligned.

```
1 .L1:
2     movupd  xmm0, XMMWORD PTR [rax+rcx]
3     add     r8, 1
4     addpd   xmm0, XMMWORD PTR [r10+rcx]
5     movaps  XMMWORD PTR [r10+rcx], xmm0
6     add     rcx, 16
7     cmp     r8, r9
8     jb      .L1
```

Listing 9.11: Assembler for the `multiply` function from Listing 9.9 with SIMD instructions.

The asm in Listing 9.10 needs eight `move` instructions and four `add` instructions to add four doubles and the asm shown in Listing 9.11 needs two `move` one `add` instructions. Intel no longer publishes cycle-count-information for the available instructions, but from benchmarks, it can be inferred that single and packed instructions require the same amount of cycles. Therefore, it is to assume that the asm in Listing 9.11 takes one forth the time to add the two arrays than the asm in Listing 9.10 [36]. This process is sometimes called vectorization.

Sometimes, the compiler will not be able to vectorize loops. It may fail to vectorize loops when the loop contains control-flow statements like the `if`-statement. Sometimes, the logic build into the compiler is just not smart enough to vectorize certain loops. If a loop cannot be vectorized, or the emitted asm is simply not fast enough, inline-assembler can be used to bypass the compiler. Inline-assembler is asm written by the programmer. It is inserted into a high-level-language, like D, and gets directly passed to the linker. Another option is to create a dedicated asm file to store the handcrafted functions. This is usually not required to write high performing programs. If functions appear to have too long runtimes, even though the programmer thought it should be vectorized or have negligible impact on the performance, it is helpful to look at the assembler output. Often, small changes to the high-level language can mitigate these problems. This is usually a trial-and-error process, as it is practically impossible or at least impractical, for the user, to understand the applied optimizations and to make educated decision how to change the high-level-code in such a way that the compiler emits better performing asm. A rule-of-thumb is to try replace the control-flow statements

with mathematical expression and to avoid loop statements where the number of loop iterations is influenced by the statements executed by the loop [47].

9.3.4 d) Link ASM

In this step, the asm generated from the different high-level language file gets linked into a single asm file and translated into binary. Modern compilers executes so called link-time-optimizations (LTO) [48]. These optimizations are done at the linking stage of the program compilation. One possible optimizations is cross-file-function-inlining. Thereby, yielding the same benefit as regular FI. FI is usually restricted to the function-definitions seen by the compiler when it compiles a particular function. For instance, let a function `fun` being defined in the file `a.d`, function `fun` calls function `bar`, and `bar` is defined in file `b.d`. Now, if the compiler compiles files `a.b` and `b.d` separately, it cannot inline `bar` in `fun`, simply because the compiler does not have access to the source code of `bar`. LTO makes inlining `bar` into `fun` possible as the linker sees the assembly created from `bar`. LTO is not such an important step for D in general. This is because the D compilers are fast enough that it is possible to pass all source files, belonging to a program, to the compiler at once. This allows the compiler to see function definitions and perform function-inlining before it comes to linking the asm. D of course comes with a standard-library. These libraries contain many functions that implement recurring tasks faced in programming. Usually, these functions can not be inlined, as only their declarations are known. As D makes heavy use of templates, nearly all functions declared in the standard-library are template-functions. This requires that their definition is visible to the compiler which allows it to inline the functions. Another common optimization at link-time is to remove unused code from the resulting binary. This results in smaller binaries which increases the cache-hit-probability. When compiling files separately, it cannot be known which symbols[3] will actually be used in the resulting binary. At link-time, this is known and can be exploited.

9.3.5 e) Translate ASM into Microcode

Intel CPU starting with the Pentium Pro in 1995 no longer execute asm directly [49]. Instead, Intel CPUs translate asm into so called micro-operations (µ-ops). These µ-ops are generally more low-level operations. For example, the `add` assembly instruction allows to directly add two values, where one of the operands can be a memory address. µ-op would translate this `add` into a `mov` and an `add` of two registers. This is represented by the Decode step in Figure 9.3. Additionally, the Pentium Pro introduced speculative execution of instructions and transparently duplicated the set of registers. The speculative execution is controlled by the scheduler as shown in Figure 9.3. The scheduler in conjunction with the Register-Alias-Table keeps track of which instruction, on which port, is currently using

[3] A symbol in this case means a label in the asm.

which register. Furthermore, the Pentium Pro gained the ability to reorder the µ-ops. These features combined is what is called out-of-order-execution. Unfortunately, Intel never published any form of notation for the µ-ops. Therefore regular asm notation is used to describe µ-ops and appropriate comments are given on the notation when required. Even though only the effects of the µ-ops are seen, an understanding of their workings helps to improve the performance of a programs. It was shown how keeping data in the L1 caches is paramount to keeping the CPU executing a program efficiently. If µ-ops can be approximated, better hypothesis of data dependencies of a program can be made, this allows to structure the program in a way such that the caches have the right data at the right time, which in turn keeps the CPU ports loaded which in turn execute the program faster.

```
1 void fun(int a, int b, int c, int* d) {
2     if(a == 0) {
3         *d = *d * (b + c);
4     } else {
5         *d = *d * (b - c);
6     }
7 }
```

Listing 9.12: A example D function.

Listing 9.12 shows a short example that will be used to demonstrate a possible execution of these instructions. The three `int` values must be unknown as otherwise the compiler might remove the `if`-statement or compute parts of the statements at CT. The pointer `*d` points to read- and write-able memory.

```
1 ; int a in [rbp+16]
2 ; int b in [rbp+12]
3 ; int c in [rbp+8]
4 ; int d in [rbp+4]
5
6 mov eax, [rbp+16]
7 cmp eax, 0
8 jne .L1
9
10 mov eax, [rbp+12]
11 add eax, [rbp+8]     ; b + c
12 imul [rbp+4]         ; * *d
13 jmp .L2
14
15 .L1
16 mov eax, [rbp+12]
17 sub eax, [rbp+8]     ; b - c
18 imul [rbp+4]         ; * *d
19
20 .L2
21 mov rcx, [rbp+4]
22 mov [rcx], eax       ; write *d back to memory
```

Listing 9.13: The example D function shown in Listing 9.12 compiled to assembler.

Listing 9.13 on the preceding page shows an possible assembler output after compilation. For the purpose of this example, the parameters are passed via the stack.

```
1  ; int a in [rbp+16]
2  ; int b in [rbp+12]
3  ; int c in [rbp+8]
4  ; int d in [rbp+4]
5
6  mov eax, [rbp+16]
7  cmp eax, 0
8  jne .L1
9
10 mov eax, [rbp+12]
11 mov ebx, [rbp+8]
12 add eax, ebx         ; b + c
13 mov rcx, [rbp+4]
14 mov edx, [rcx]
15 imul edx             ; * *d
16 jmp .L2
17
18 .L1
19 mov eax, [rbp+12]
20 mov ebx, [rbp+8]
21 sub eax, ebx         ; b + c
22 mov rcx, [rbp+4]
23 mov edx, [rcx]
24 imul edx             ; * *d
25
26 jmp .L2
27 mov rcx, [rbp+4]
28 mov [rcx], eax       ; write *d back to memory
```

Listing 9.14: The assemble of the example function being transformed into µ-op.

After the binary file has been loaded by the CPU, it transforms the binary into µ-ops. One attribute of µ-op is that other than load and store operations all operations exclusively work on registers. That means that operations like `add eax , [rbp+16]` are no longer possible and have to be translated into a `mov` and `add`. In order to visualize this process, the transformation in asm is shown in Listing 9.14.

```
1  ; int a in [rbp+16]
2  ; int b in [rbp+12]
3  ; int c in [rbp+8]
4  ; int d in [rbp+4]
5
6  mov eax_1, [rbp+16]
7  cmp eax_1, 0
8  jne .L1
9
10 mov eax_2, [rbp+12]
11 mov ebx_2, [rbp+8]
12 add eax_2, ebx_2     ; b + c
13 mov rcx_2, [rbp+4]
14 mov edx_2, [rcx_2]
```

```
15 imul eax_2, edx_2    ; * *d
16 jmp .L2
17
18 .L1
19 mov eax_3, [rbp+12]
20 mov ebx_3, [rbp+8]
21 sub eax_3, ebx_3     ; b + c
22 mov rcx_3, [rbp+4]
23 mov edx_3, [rcx_3]
24 imul eax_3, edx_3    ; * *d
25
26 .L2
27 mov rcx_4, [rbp+4]
28 mov [rcx_4], eax_?   ; write *d back to memory
```

Listing 9.15: µ-op allowed to use multi-register-sets.

As mentioned earlier, modern CPUs have multiple sets of registers. This simplifies the use of the multiple arithmetic-logical-units (ALUs) present in modern CPUs. In this example, the utilization of these multiple-register sets is also done in this stage. The different register sets are distinguished by the suffix. The suffix `_1` points to the first register set, the suffix `_2` points to the second and so on. The goal here is to use as many of the register sets as possible. The result of this transformation is shown in Listing 9.15 on the facing page. The last line of Listing 9.15 reveals a problem. Writing back a value to memory does make it potentially visible[4] by other programs, but the two competing results are stored in `eax_2` and `eax_3`. The question becomes, which of those values needs to be written to the memory location `[rcx_4]`. This is solved in Subsection 9.3.8 on page 218.

9.3.6 f) Reorder Microcode Instructions

As mentioned earlier, the goal of the microcode is to utilize all the CPU resources as much as possible. For that to work, the CPU needs data, but communicating date from and to memory is slow, too slow to fully utilize the CPU. Even with caches, getting data may be too slow. To combat that, the CPU reorders the µ-ops such that all load instruction happens as soon as possible.

```
1 ; int a in [rbp+16]
2 ; int b in [rbp+12]
3 ; int c in [rbp+8]
4 ; int d in [rbp+4]
5
6 mov eax_1, [rbp+16]
7 mov eax_2, [rbp+12]
8 mov ebx_2, [rbp+8]
9 mov eax_3, [rbp+12]
10 mov ebx_3, [rbp+8]
11 mov rcx_2, [rbp+4]
```

[4] Visible, in this case means, another process could read the value if they had access to the memory location.

```
12 mov edx_2, [rcx_2]
13 mov rcx_3, [rbp+4]
14 mov edx_3, [rcx_3]
15 mov rcx_4, [rbp+4]
16
17 cmp eax_1, 0
18 jne .L1
19
20 add eax_2, ebx_2    ; b + c
21 imul eax_2, edx_2   ; * *d
22 jmp .L2
23
24 .L1
25 sub eax_3, ebx_3    ; b + c
26 imul eax_3, edx_3   ; * *d
27
28 .L2
29 mov [rcx_4], eax_?  ; write *d back to memory
```

Listing 9.16: The assemble of the example function reordered.

Listing 9.16 on the preceding page shows this transformation. All memory loads operations are at the beginning of the shown listing.

µ-ops are also allowed to rewrite the binary, this is shown in Listing 9.17.

```
1 ; int a in [rbp+16]
2 ; int b in [rbp+12]
3 ; int c in [rbp+8]
4 ; int d in [rbp+4]
5
6 mov eax_1, [rbp+16]
7 mov eax_2, [rbp+12]
8 mov ebx_2, [rbp+8]
9 mov eax_3, [rbp+12]
10 mov ebx_3, [rbp+8]
11 mov rcx_2, [rbp+4]
12 mov edx_2, [rcx_2]
13 mov rcx_3, [rbp+4]
14 mov edx_3, [rcx_3]
15 mov rcx_4, [rbp+4]
16
17 cmp eax_1, 0
18
19 add eax_2, ebx_2    ; b + c
20 imul eax_2, edx_2   ; * *d
21 cmove eax_4, eax_2
22
23 sub eax_3, ebx_3    ; b + c
24 imul eax_3, edx_3   ; * *d
25 cmovne eax_5, eax_3
26
27 mov [rcx_4], eax_?  ; write *d back to memory
```

Listing 9.17: The assemble of the example function reordered continued.

The `cmp` and `jmp` control-flow is replaced with `cmove` and `cmovne` operations. `cmove` stands for conditional-`mov`-equal and only moves the right operand to the left operand if the previous compare-operation yielded the value `true`. `cmovne` stands for conditional-`mov`-not-equal and only moves the right operand to the left operand if the previous compare yielded the value `false`. This transformation will help significantly in the following step. Sometimes, the compiler is not able to create asm that is conducive to these kinds of reorderings. The µ-ops cannot be seen directly, but the program perf can be used to observe stall cycles. Stall cycles give information how often the CPU doesn't have µ-ops ready to execute or is missing resources. Stall cycles are used to spot problems in the µ-ops programs. Reordering the D-code and replacing `if`-statements with tenary-operations showed good results in reducing the stall cycle count and thereby improve CPU utilization.

9.3.7 g) Execute Microcode instructions

For the example, execution of the µ-ops as shown in Listing 9.17, the following state of the CPU, memory, and caches is assumed. The memory of the addresses `rbp+12`, `rbp+8`, `rbp+4`, and `[rbp+4]` are in the L1 cache of the CPU. The memory of the variable `int a` is currently not cached. For the purposes of demonstration, the ALUs are currently not used. Additionally, it is assumed that moving data from the L1 cache to a register takes one time unit, moving data from memory to a cache or vice-versa takes five time units, the CPU can start the execution of four lines of µ-op per time unit, the CPU can start one move from memory to the cache in each time unit, all other operations take one time unit to execute. The eight time units below show the start and the finish time of each instruction.

1. `mov eax_1, [rbp+16]` started
   `mov eax_2, [rbp+12]` completed
   `mov ebx_2, [rbp+8]` completed
   `mov eax_3, [rbp+12]` completed

2. `mov ebx_3, [rbp+8]` completed
   `mov rcx_2, [rbp+4]` completed
   `mov edx_2, [rcx_2]` completed
   `mov rcx_3, [rbp+4]` completed

3. `mov edx_3, [rcx_3]` completed
   `mov rcx_4, [rbp+4]` completed
   `add eax_2, ebx_2` completed
   `sub eax_3, ebx_3` completed

4. `imul eax_2, edx_2` completed
   `imul eax_3, edx_3` completed

5. `mov eax_1, [rbp+16]` completed

6. `cmp eax_1, 0` completed
7. `cmove eax_4, eax_2` completed
   `cmovne eax_5, eax_3` completed
8. `mov [rcx_4], eax_?` started

9.3.8 h) Retire Microcode Instructions

In the last stage, the CPU figures out which of the `eax` values it needs to write to memory or to the permanent register file. This is where the out-of-order execution is made "in-order" again, in the sense that the observable effect done by the out-of-order execution is the same as the effect that would have been observed by in an in-order execution [36].

9.4 Optimizations

Now that the CPU used to execute the analysis program has been introduced, concrete optimizations will be presented.

9.4.1 The graph structure Data structure

As mentioned before, the caches are an important factor when it comes to program performance. The data-structures (DSTs) used to store the data of a program directly influence how data is stored in RAM and, therefore, indirectly influence how data is stored in the caches. DSTs are therefore very important, when it comes to cache utilization. At the core of MC are GSs. GSs are used to represent LNTs and PNTs. LNTs and PNTs are used in the mapping approach (MA), the CIP, and the CP. There are two common ways for DSTs to store GSs. The first one is shown in Listing 9.18.

```
1 class Vertex {
2     int id;
3     vec2f position;
4     Vertex[] adjacent;
5 }
6
7 struct GS {
8     Vertex[] vertices;
9 }
```

Listing 9.18: A DST to store a GS.

This DST has several disadvantages. The `class Vertex` is heap-allocated. Therefore, the program has no control where in memory a vertex is stored. This decreases the chance of cache-hits. As mentioned previously, cache-hits play a important role for the performance of a program. Additionally, the `class Vertex` stores its adjacent vertices in a dynamic array. This means for each access to any

of the adjacent vertices of a `Vertex` the CPU has to follow two pointers. One for the memory of the array and one for the `Vertex`. This introduces two chances of cache-misses. The size of one instance of the `class Vertex` is 28 bytes, ignoring default class members like the vtable [50]. The `struct GS` has a size of 16 bytes. Another problem is that removing edges is computational complex. One of the vertices creating the edges has to be found in their `vertices` array of the `struct GS`. As it cannot guaranteed that the vertices are sorted by their `id` in this array the complete array needs to be searched. This entails a $\mathcal{O}(N)$ complexity where N is the number of vertices in the GS. Then, the second vertices in the array called `adjacent` has to be found. Again, this has a $\mathcal{O}(N)$ complexity. As it cannot be assumed that the `Vertex` to remove is at the end of the array, the program might have to remove the `Vertex` from an arbitrary position in the array. This again, entails a $\mathcal{O}(N)$ complexity. As the GS is not a directed GS, this procedure has to be repeated for the second vertex.

Testing whether there is an edge between two vertices is also unnecessarily complex. For this procedure, the program has to find the first `Vertex` in the `vertices` array and then the second `Vertex` in the adjacent array. Both searches have a $\mathcal{O}(N)$ complexity.

The DSTs shown in Listing 9.19 mitigates some of the previous problems. The shown DSTs are another common way of representing the GS.

```
1 class Vertex {
2     int id;
3     vec2f position;
4 }
5
6 struct GS {
7     Vertex[] vertices;
8     int[][] edges;
9 }
```

Listing 9.19: A optimized DST to store a GS.

There are still performance problems and inefficacies with this design. Due to experiments, it is known that no tests with more than 32 vertices are run. That means, if an `int` encodes the vertex IDs, the program wastes at least 27 bits of each `int`. The edge lookup and removal still requires multiple, linear searches through the array storing the `edges`.

Ideally, inserting, removing, and testing edges should have constant runtime complexity. Additionally, the DSTs used to store the GS should be as cache friendly as possible. Looking closer at the hot spots of MC, revealed that the path finding procedure was highly critical. That means, the edge data is required to be rapidly accessible, the position data of the vertices on the other hand was not as important.

Listing 9.20 shows a simplified template `struct` of the used GS.

```
1 struct GS(T) {
2     T[T.sizeof * 8] edges;
```

```
3     ubyte numberVertices;
4
5     vec2f[T.sizeof * 8] positions;
6 }
```

Listing 9.20: A template `struct` used to store the GSs in MC.

`T` is the template type of the `struct`. The statement `GS!ubyte` will create a `struct` where each occurrence of `T` is replaced with `ubyte`. The type `ubyte` is an `unsigned int` type.

```
1 struct GS!(ubyte) {
2     ubyte[8] edges;
3     ubyte numberVertices;
4
5     vec2f[8] positions;
6 }
```

Listing 9.21: The template shown in Listing 9.20 instantiated with the type `ubyte`.

The DST in Listing 9.21 can, of course, store GSs with less than nine vertices but for some analyses this is sufficient. All the data of the DST is aligned consecutively in memory and can be accessed in constant time without following pointers. At first glance, the `edges` array looks insufficient to store all the edges, but it is not. Now it is made sure that the ID of a vertex is equal to its position in the array. Meaning IDs start with the number zero, and all IDs are consecutively. The adjacency list is stored as a bit-mask. For the analyses done in this work, it is only required to know whether an edge exists or not. The length of the edge is not relevant. Therefore, a boolean type is sufficient to store it. D offers a boolean type in the form of `bool`, but this type requires one byte to store it. This would mean that always 7 bits are wasted. With bit operations, in case of the template instantiation shown in Listing 9.21, the eight bits of each element of `edges` can be addressed individually. A bit set to 1 means the edge exists, a bit set to 0 means no edge exists. To test if an edge $e_{1,3}$ exists, MC just has to evaluate the expression `edges[1] & (1U << 3)`[5]. If this expression evaluates to `true` the edge exists, otherwise it does not. Testing the existence of the edge, starting from the other direction, is equally as easy. Creating a new edge is not much harder as shown below. Creating the edge $e_{2,4}$ is done by the two statements:

```
1     edges[2] = edges[2] | (1U << 4);
2     edges[4] = edges[4] | (1U << 2);
```

The size of `GS!(ubyte)` is 73 byte, which the compiler extends to 80 to make it align better into memory. Compared to the 28 bytes for a single vertex, without any edges, for the `class` shown in Listing 9.18 this is an substantial reduction of required memory. This means the complete `GS!(ubyte)` DST will not fit into one L1 cache-line. Luckily, the adjacency array in combinations with the size of the

[5] `1U` is D shorthand to create an `uint` with value 1.

GS fits comfortably, as it only requires nine bytes and data is loaded in cache-line sized blocks. This means when a GS DST specialized to `ubyte` is loaded, the complete adjacency array, the length, and part of the position vectors are loaded [6]. Such a DST limits the size of the analyzed GS to eight vertices as an `ubyte` can only represent eight boolean values.

```
1 struct GS!(ushort) {
2     ubyte[16] edges;
3     ubyte numberVertices;
4
5     vec2f[16] positions;
6 }
```

Listing 9.22: The template shown in Listing 9.20 instantiated with the type `ushort`.

Listing 9.22 shows the template shown in Listing 9.20 instantiated with a `ushort`. Such a DST limits the size of the stored GS to 16 vertices, which is enough for most of the analyses conducted in this work. In D `ushort`s are always stored in two bytes. The resulting size of the `struct` shown in Listing 9.22 is therefore, 168 bytes. The adjacency array and length of the GS DST is 40 byte. This still fits into a single L1 cache-line.

This reduction in size played an important role in decreasing the runtime of each analysis, as this optimization drop the time a mapping required from months to many weeks.

9.4.2 Quorum Protocol Based Optimizations

For a mapping of a QP with N replicas $N!$ mappings have to be calculated. With growing N such analyses quickly become infeasible. Reducing the number of mappings required for the analysis of a mapping would allow to greatly increase the scope of the possible analyses. Ideally, testing only a single mapping would lead to the optimal mapping. For the MCS, this is possible. Due to the symmetric structure of the resulting RQS and WQS, all mappings will result in mapped RQS and mapped WQS. Consider the RQS of the MCS with four replias as shown in Table 9.1 on the next page. Given the PNT shown in Figure 9.5 and the mapping $a = \{(0,a),(1,b),(2,c),(3,d)\}$, the result is mapped RQS as shown in Table 9.2 on the next page. If a mapping $a = \{(0,b),(1,a),(2,d),(3,c)\}$ is used the result is the

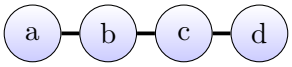


Figure 9.5: A PNT used to show the symmetric nature of mappings of the MCS.

RQS as shown in Table 9.3 on the following page. Comparing the two mapped RQSs shown in Table 9.2 and Table 9.3, shows that the two mapped RQSs have the same structures and only the individual q_i and $sq_{i,j}$ elements are only named

[6] This is also the reason why ordering the elements of a `struct` or `class` by their access frequency can have significant performance implications.

$$
\begin{aligned}
\text{RQS} = \{ & \\
& (\{0,1\},\{\{0,1,2\},\{0,1,3\},\{0,1,2,3\}\}), \\
& (\{0,2\},\{\{0,2,3\}\}), \\
& (\{0,3\},\{\}), \\
& (\{1,2\},\{\{1,2,3\}\}), \\
& (\{1,3\},\{\}), \\
& (\{2,3\},\{\}), \\
& \quad \}
\end{aligned}
$$

Table 9.1: RQS of the MCS with four replicas.

$$
\begin{aligned}
\text{RQS} = \{ & \\
& (\{a,b\},\{\{a,b,c\},\{a,b,c,d\}\}), \\
& (\{b,c\},\{\{b,c,d\}\}), \\
& (\{c,d\},\{\}) \\
& \quad \}
\end{aligned}
$$

Table 9.2: The mapped RQS of the MCS with four replicas with the mapping $\{(0,a),(1,b),(2,c),(3,d)\}$.

$$
\begin{aligned}
\text{RQS} = \{ & \\
& (\{b,a\},\{\{b,a,c\},\{b,a,c,d\}\}), \\
& (\{b,c\},\{\{b,c,d\}\}), \\
& (\{c,d\},\{\}) \\
& \quad \}
\end{aligned}
$$

Table 9.3: The mapped RQS of the MCS with four replicas with the mapping $\{(0,b),(1,a),(2,d),(3,c)\}$.

differently. Realizing this, allows to MC to test only one MCS mapping to find the optimal mapping.

9.4.3 Read Quorum Set and Write Quorum Set Construction

During analyses of the mapping approach implementation it became clear that the use of the floyd-warshall path finding algorithm (FWA) [51] had significant, negative impact on the performance. Path finding is at the heard of reconstructing communication paths for mappings. Even though the FWA was significantly more efficient in this use-case than, for instance the Dijkstra's algorithm [52] or the A* [53], it was still too expensive to call. The goal was therefore to reduce the amount of executions of the FWA.

The definitions of the RQS Equation 3.5 on page 16 and the WQS Equation 3.15 on page 16 contain some interesting insights. Originally, each combination of vertices was tested whether they are a RQ or a WQ. This is true for some QPs and for all mapped QPs.

The q_i elements always contain less vertices than their $sq_{i,j}$ elements and additionally they are supersets of the q_i elements. If a set of vertices a is a superset of q_i and q_i is a RQ (or a WQ), then a is also a RQ (or a WQ). Testing whether a is a superset is a lot faster than testing whether a is a RQ or a WQ. Listing 9.23 shows a function testing if `b` is a superset of `a`.

```
1 bool isSuperSet(int[] a, int[] b) {
2     outer: foreach(it; a) {
3         foreach(jt; b) {
4             if(it == jt) {
5                 continue outer;
6             }
7         }
8         return false;
9     }
10    return true;
11 }
```

Listing 9.23: A function that checks if `b` is a superset of `a`.

A implementation of such a test can be seen in function `isSuperSet` in Listing 9.23. This function tests whether all the elements of the array `a` are in `b`. This function has a runtime complexity of $\mathcal{O}(N)$.

In order to build the complete RQS (and WQS) the program has to iterate all combinations of vertices assumed to be available. If the program starts with the one-element sets, continue with the two element sets and so on, it can first test for the superset property and see whether the current set is a superset to an already found RQ or WQ. If there is a matching quorum it simply stores the current set as a superset. If it does not find a matching subset it tests whether the current set is a RQ (or WQ respectively). If it is, it is stored as a q_i element. If it is assumed the DS used to build the RQS and WQS is shown in Listing 9.26 on page 225, the

function shown in Listing 9.25 on the next page can be used to build the RQS or the WQS.

```
1 struct Q_SQ {
2     int[] qi;
3     int[][] sqij;
4 }
5
6 struct RQS {
7     Q_SQ[] elements;
8 }
```

Listing 9.24: A example DS to build a RQS

```
1  RQS buildRQS(int n) {
2      RQS rqs;
3      for(int i = 0; i <= n; ++i) {
4          outer: foreach(int[] it; nElementSets(i, n)) {
5              // test if `it` is a superset to any of the elements in
                   the
6              // currently build rqs
7              foreach(Q_SQ qsq; rqs.elements) {
8                  if(isSuperSet(qsq.qi, it)) {
9                      qsq.sqij ~= it;
10                     continue outer;
11                 }
12             }
13         }
14
15         // test if `it` is a read quorum
16         if(isReadQuorum(it, n)) {
17             Q_SQ tmp;
18             tmp.qi = it;
19             rqs.elements ~= tmp;
20         }
21     }
22     return rqs;
23 }
```

Listing 9.25: The function used to build the RQS.

The variable `rqs` is DS used to store the RQS. The loop in line 3 in Listing 9.25 iterates the variable `i` from 1 to including $|V(g)|$ in order where `g` is a GS used by the QP. The following `foreach` loop in line 4 uses a helper function called `nElementsSets`, that creates all sets with `i` elements out of a set of `g.length` elements. On line 7 in Listing 9.25 the search for the superset condition is started, `qsq` is the iterator that walks front to back through the $(q_i, \{sq_{i,0}, \ldots sq_{i,j}\})$ elements of the `rqs`. If `it` is a superset of `qsq.qi`, `it` is appended to the set of supersets of `qsq` as shown in line 9. The operator `~=` of D appends an element to an array. As there is nothing more to do with `it`, the program can continue with the next set. This is accomplished by the `continue outer` in line 10. The statement `continue outer` has the same semantic as the know `continue` statement in all other C-based languages, expect that it

continues the labeled loop and not the enclosing loop. If no superset relation can be found, then the function tests whether `it` is a RQ as shown in line 16. If that is the case a new `Q_SQ` element is created and stored in `rqs`. After all sets of elements have been tested, `rqs` is returned, thereby concluding the RQS construction. This function exploits a little different mechanism to decrease the construction time of the RQS. As the sets are tested from few to many elements, the function can often skip the, possible expensive, QP logic and solely depend on set operations to determine if a set is a RQ. This works, because it is assumed that a QP will always use the smallest possible quorum, even if additional replicas are available, as shown in Equation 3.14 on page 16. Due to the construction of the `elements` array in `RQS` RQs with few elements are stored at the beginning of the array. The search for a superset is also started at the beginning of the array. It is therefore likely that for most RQs a superset relation is found early in the search. This has been tested by instrumenting code to count how often what path has been taken. For most QPs the process of calling the `buildRQS` function executed as follows: Initially, the actual `isReadQuorum` function is executed multiple times to build few minimal quorums. After these changes, most new quorums were inserted without resorting to the expensive `isReadQuorum` function.

This can be illustrated by the MCS. Let `buildRQS` be called with `n` equal to five. After the ten distinct three element RQs have been inserted into the `rqs`, the six additionally tested sets will be successfully evaluated by the superset branch of the function.

The function `buildRQS` shown in Listing 9.25 still has some potential for improvement. The `foreach` in line 4 in Listing 9.25 creates a new array for each iteration. As it has been previously discussed, arrays are to be avoided to increase the number of cache hits. As the program has control over how replicas are identified, MC identifies N replicas with the IDs from 0 to $N-1$. This way, MC can store the available replicas of a QP with up to 32 as a bit mask inside a single `uint`. As the extent of the analyses is known a priori, the user can select an appropriate type at CT. This allows MC to use an improved GS as shown in Listing 9.26. The `int` array previously used to store the RQ was replaced by a `uint`. The subsets are now stored in a one-dimensional `uint` array.

```
1 struct Q_SQ {
2     uint qi;
3     uint[] sqij;
4 }
5
6 struct RQS {
7     Q_SQ[] elements;
8 }
```

Listing 9.26: A improved DS used to build a RQS

This allows MC to make some significant improvements in the improved `buildRQS` function shown in Listing 9.27.

```
1 RQS buildRQS(int n) {
```

```
2     RQS rqs;
3     for(int i = 0; i <= n; ++i) {
4         outer: foreach(uint it; nElementSets(i, n)) {
5             // test if `it` is a superset to any of the elements in
                  the
6             // currently build rqs
7             foreach(Q_SQ qsq; rqs.elements) {
8                 if((qsq.qi & it) == qsq.qi) {
9                     qsq.sqij ~= it;
10                    continue outer;
11                }
12            }
13        }
14
15        // test if `it` is a read quorum
16        if(isReadQuorum(it, n)) {
17            Q_SQ tmp;
18            tmp.qi = it;
19            tmp.sqil.reserve(16);
20            rqs.elements ~= tmp;
21        }
22    }
23    return rqs;
24 }
```

Listing 9.27: A improved function used to build the RQS.

The first improvement is shown in line 4 in Listing 9.27. Instead of returning an array for each iteration, `nElementsSets` returns a single `uint`. This removed the need for memory allocation and the values can be stored on the stack, increasing the cache-hit change further.

The largest improvement comes from the changed superset test. Instead of potentially comparing all elements of two elements, a single bitwise `and` operation combined with a comparison is used. Consider the set $a = \{0,2,3,5\}$ and the set $b = \{0,2,3,4,5,7\}$. Table 9.4 shows the result of the bitwise `and` operation of these two sets represented as bitmasks and the following comparison. The example

$\{0,2,3,5\}$	=	0	0	1	0	1	1	0	1	
$\{0,2,3,4,5,7\}$	=	1	0	1	1	1	1	0	1	
&	=	0	0	1	0	1	1	0	1	$= \{0,2,3,5\}$

Table 9.4: Example of a successful superset test with two sets with the bitwise and operation.

shows how the IDs stored in the set are used to set the bits in the bitmask and how the binary and operation combines the two bitmasks. The result is then transformed into a regular set again for easy comparison between sets. As the example shows, the result of the bitwise and is equal to the set a. Table 9.5 on the facing page shows an example where $b = \{0,2,3,4,5,7\}$ is not a superset of $a = \{0,2,3,5\}$. Here the result is not equal to a.

$\{0,2,3,5\}$	=	0	0	1	0	1	1	0	1	
$\{0,2,4,5,7\}$	=	1	0	1	1	0	1	0	1	
&	=	0	0	1	0	0	1	0	1	$=\{0,2,5\}$

Table 9.5: Example of an unsuccessful superset test with two sets using the bitwise and operation.

The last small, notable improvement was in the memory management for the superset array. In line 19 in Listing 9.27 a 16 element is allocated for the superset array. This is done to reduce the number of expected allocations of the span of the execution of the function. The first few RQs will likely have many supersets. This way the first 16 can be stored without any memory management overhead. The size of 16 was chosen, because 16 `uint`s fill exactly one L1-cache-line, further increasing the chance for cache hits. This superset search is used for all QPs discussed in this work.

The construction of the WQS is analogous to the RQS construction shown.

The RQS of the QP mapped, and a DS representing the PNT is used as input into the function. The superset test is also applied, but instead of testing whether some set is a RQ, it is tested by the elements of the set are connected in the given PNT.

With these performance optimizations the execution time for a mapping with nine replicas was down to several hours and therefore the termination criterion of the optimization process shown in Figure 9.1 was reached.

9.5 Conclusion

This chapter showed the overall process used to optimize MC.

Additionally, a very quick overview was given how modern CPU actual work. This knowledge was then exploited to increase the performance of MC to make the analysis of the mapping approach, as well as the CIP and the CP, feasible.

Furthermore, two rather low level optimizations were explored to give an impression of the depth of the required optimizations.

Due to the work presented in this chapter, the analysis time of mapping of the TLP with nine replicas to a PNT was reduced from seven month to about seven hours[7].

9.6 Future Work

Currently, the performance of MC is restricting the analyses of the mapping approach to QPs with nine replicas. The next target would be to test the mappings

[7]The seven month timespan is an extrapolation based on a seven day execution of the MC program that had to be interrupted due to obvious reasons.

of QPs with 12 replicas. This would allow to analyze the TLP with a 3×4 LNT or the GP with a similar LNT. A mapping of 12 replicas is 1320 more complex a mapping with nine replicas, due to the factorial runtime complexity of the mapping approach. Assuming no process loss and enough memory, this would require about a year of computing, making it infeasible for the scope of this work. PNT with ten or eleven vertices are not that interesting as the possible LNT for the TLP and the GP either 2×5 or 1×11 which are heavily unbalanced towards read operation. The 1×11 grid would even result in the QP and TLP into the Read-One/Write-All (ROWA) QPs.

9.6.1 Multithreading

Currently, all analyses are executed on a single CPU core. The utilization of multiple CPU cores are achieved by executing MC multiple instances per CPU. This allows the different instances of MC to work independently, without the need for synchronization.

If no optimization like the one shown in Subsection 9.4.2 are found for the analyzed QPs a multithreaded analysis will be a logical next step. This means that the analysis is spread among multiple CPU-cores. This introduces the need for synchronization between the executed threads.

Creating the RQS and the WQS multithreaded requires extensive synchronization. The synchronization would likely be inside the DST used to store the RQS or the WQS, respectively. The problem with multithreading is the performance of the synchronization primitives offered by the CPUs. Atomic writes operations are up to 30 times slower than regular writes, and additionally the mutations of the shared DS have to be done in a synchronized manner [54]. This requires locking, which might stop the execution of other threads.

Parallelizing the mapping approach is easy as most computation can be done without interaction between threads. Such a parallelization would likely map one mapping to one thread. After the ARW of a mapping has been finally computed, it is tested whether this mapping has the currently highest ARW. If that is the case, this mapping is stored as the currently best mapping. This ARW comparison, and the possible storage of the mapping has to be done in a synchronized manner. Depending on the number of available CPUs, this could reduce the computation time significantly. Such a approach is known as a map-reduce process [55].

9.6.2 Rule based RQS and WQS construction

The RQS and the WQS are constructed in the same way for each tested QP. The only difference is the test used to determine whether a set of replicas is a RQ or a WQ. This is not optimal in the general case. Optimal, here means that it requires the least possible amount of computation.

Consider an MCS with five replicas. No set with less than three replicas can form a RQ or a WQ. With this knowledge, the program can exclude 15 of the 31 sets that would normally tested.

Such optimization might exists for other QPs as well. Searching for these optimizations might be a worthwhile task.

10 Conclusion

This work has shown how strong some of the assumptions of existing QPs are, and what big difference weaker, and therefore more realistic, assumptions make when analyzing QPs. This was done be the introduction of the LNT, PNT, and the mapping approach. This approach was used to extensively analyze different QPs and PNT combinations.

Due to the computational complexity of the mapping approach, the kNN method was successfully used to predict the influence of a mapping based on features extracted from the used PNT. Here, it was shown that the BC_{max} feature is a good indicator to predict the cost and availability measurements of a given PNT. Additionally, a was found to reduce the number of mappings that have to analyzed for the MCS from $N!$ to 1, where N is the number of replicas used by the MCS.

As the existing QPs where not meant to mapped to more realistic GSs, the read and write availability, and the cost measures degenerated so much that QPs had to be introduced that directly applied to a given PNT. Therefore, the CIP, a new QP, was introduced. The main advantage of the CIP is that it does not require any specific LNT, instead it directly works on the given PNT. This allows it to be applied to PNTs with much more vertices than what is currently feasible using the mapping approach. It was shown how the CIP can use many PNTs, and how PNTs need to be prepared in order to be used by the CIP. As the structure of the PNT is considered unchangeable by this work, it was explained why those preparations, required by the CIP, do not undermine this requirement.

Based on the experiences and results of the CIP, another QP, called CP, was developed. Similarly to the CIP, the CP directly works on a given PNT. It was developed to combine the best properties of the CIP and the TLP. This worked for the most part, but unfavorable PNTs still showed limits to this new QPs. With larger QP the CIP results became better as the planarization step no longer resulted in GS that mirrored lines.

As it became clear that no one QP was always the best, a new algorithm was developed that given a PNT and a set of QPs, finds the best possible QP for a given PNT. This algorithm works for all QPs for which a RQS, and WQS can be constructed. These two concepts where introduced and used throughout this work to analyze all QPs.

Many of the analyses performed were computational extremely complex, so much so that computing even partial results would have taken weeks or months. To make any meaningful statement about the various QPs as well as the presented approaches and algorithms, the execution speed of MC used for the analyses had to be improved significantly. An excursion, into some of the used optimizations

techniques was present as well as introduction into modern CPU architectures and there implication for software performance.

Finally, some ideas for possible future works building on concepts introduced in this week are presented.

11 Future Work

During this work more ideas were had and more research was done. This chapter is an overview of the most promising ones.

11.1 Evaluation of more Quorum Protocols

In this work, three known QPs, and two new QPs where evaluated, but there are many more QPs. Some of the better known are the Generalized Tree QP [56] and the Crumbling Walls QP [57]. Adding these QPs to MC and evaluating their cost and availability measurements, when mapped to different PNTs, would give an even deeper insight into the performance of the mapping approach. Additionally, these additions would allow to choose from a greater set of QPs in Algorithm 20. This was not done as it would not have added something conceptually new.

11.2 Evaluation of Additional Graph Structure Features

Even though the prediction accuracy of the kNN approach is already very good, many GS features remain untested. The prediction accuracy of the BC feature works notably good, but it is also the computationally most complex GS feature analyzed. Analyzing more features might yield features that are even better suitable for predictions.

One possible candidate is the Circuit rank [58]. The Circuit rank describes how many edges of a GS need to be removed to transform it into a tree. Another candidate that might be worth analyzing is what is known as cliques [59]. Clique describes how many vertices form completely connected subgraphs of a GS.

As the BC_{max} already had a MSE of 0, the optimal MSE, this work was not deemed necessary.

11.3 Decreasing the Analysis Time

Analyzing the $a_r(p)$, the $a_w(p)$, the $c_r(p)$, and the $c_w(p)$ of a mapping of the TLP with nine replicas mapped to a PNT, also with nine replicas, takes about seven hours on a somewhat modern CPUs. Increasing the number of vertices of the analyzed LNT and PNT to even only ten replicas would increase the analysis time to about 70 hours. Decreasing the time required for an analysis, or even developing an analysis technique with less than factorial complexity, would allow to study the mapping approach in more detail.

11.4 Consider Edge Availability

Currently, edges are assumed to have a p-value of 1.0, meaning they are always available. Obviously, this is a simplification whose removal would decrease the difference between analysis and the real-world behavior. This would in turn increase the analysis time in an unreasonable about, which for even very small GSs, less than five replicas, would likely exceed many weeks. This is because the complexity would grow factorial.

11.5 Consider Vertices without Replicas

In this work all vertices of a GS host exactly one replica. Allowing vertices to host zero or more replicas would allow to further decrease the gap between LNTs and PNTs. For instance, having a star like GS, as shown in Figure 11.1, with the center vertex 0 not hosting a replica would allow to evaluate very common real-world network structures.

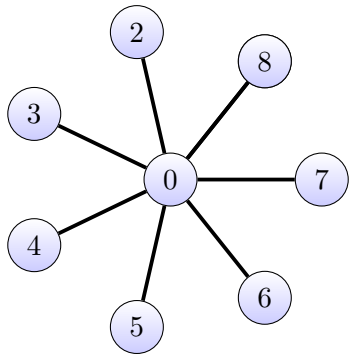


Figure 11.1: Star like GS.

11.6 Individual p-Values

The p-value of all replicas are currently considered equal. This is mostly a simplification to make analyses feasible. Lifting this assumption would allow to create evaluations of QPs and mappings that more closely model real-world scenarios. Again, the increase in analysis time would make such an approach infeasible.

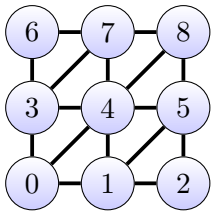
11.7 Graph Structure Planarization

The way GSs are currently planarizied leaves many GSs unusable for the CIP as well as the CP. If possible, this should be avoided. One possibility to remove many non-planar structures from a GS is to move the vertices to different positions in the plane. An instance of this is shown in Figure 6.7 and Figure 6.8 on page 134.

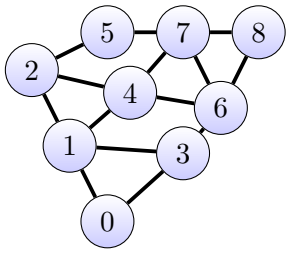
Planarizing such a GS by exhaustively searching for the best move operations becomes extremely computational complex, if not infeasible. To make it feasible, applying heuristics, such as the ones presented in [60] and [61], could be used.

11.8 Approximating Mappings

Testing all $\mathcal{O}(N!)$ possible mappings, where N is the number of replicas, to find the optimal mapping does not scale well. Figuring out which replica of a given LNT should be mapped to which replica of the PNT is the central question of the mapping approach. To find the optimal mapping, currently, all possible mappings have to be evaluated. One idea to find not necessarily the optimal, but a good mapping might be to assume replicas hosted by specific vertices of the LNT to be mapped to vertices that have similar features on the PNT. Therefore, all the



(a) The LNT used the experiment to approximate the optimal mapping



(b) The LNT used the experiment to approximate the optimal mapping

Figure 11.2: GS used in an experiment to approximate the optimal mapping

vertices of, both, the LNT as well as the PNT are sorted by some feature or tuple of features. Table 11.1 shows the sorting of the vertices of the GSs in Figure 11.2b and Figure 11.2b sorted by the (Dia_{Avg}, BC) features. Now that

LNT		PNT	
ID	(Dia_{Avg}, BC)	ID	(Dia_{Avg}, BC)
4	(2.25,8)	4	(2.5,5)
8	(2.625,0)	6	(2.5,6)
7	(2.625,1)	7	(2.625,2)
5	(2.625,1)	1	(2.625,7)
0	(2.625,4)	2	(2.75,4)
1	(2.625,6)	3	(2.75,4)
3	(2.625,6)	5	(3,0)
2	(3.25,0)	0	(3.125,0)
6	(3.25,0)	8	(3.125,0)

Table 11.1: Sorting of the vertices of GSs in Figure 11.2b and Figure 11.2b by the BC feature.

the vertices are sorted, a mapping can be constructed as follows. The vertex of the LNT with ID 1 is mapped to the vertex of the PNT with ID 1. The vertex with ID 3 is mapped to the vertex with ID 6, and so forth. This results in the mapping $\{(1,1),(3,6),(4,7),(5,4),(7,0),(0,2),(2,3),(6,5),(8,8)\}$. Figure 11.3a on the next page shows the $a_r(p)$ and $a_w(p)$ of the TLP (TLP-3x3), the best mapping (BestMap), and the mapping resulting from the sorting of the vertices (SortMap). The $a_r(p)$ of SortMap is by no means comparable to the $a_r(p)$ of BestMap, the $a_w(p)$ on the other hand is extremely similar to the $a_w(p)$ of BestMap, even surpassing it in the low p-value range. Figure 11.3b on the facing page shows the $c_r(p)$ and the $c_w(p)$. Here the SortMap mapping performs similar to the BestMap mapping. So far, only the above shown LNT and PNT where tested with the features, Minimum Diameter, Maximum Diameter, Average Diameter, Mode Diameter, Median Diameter, Degree, BC, and all two element combinations of the previously mentioned features. From these tests, the combination (Average Diameter, BC) had the highest ARW with 90.92, which is close to the ARW of the best mapping with 94.65 [rsot17long].

Investigating more feature combinations with more QPs and PNTs might allow to build a heuristic to approximate the best mapping with significant less computational complexity. This approach is especially promising with larger GSs, but in order to achieved operation availabilities and costs comparable the optimal mappings still would have to be found, which is again computational extremely expensive.

Concluding this work, it is obvious that the assumptions about the analyzed systems are extremely important, but with the approaches presented in this work theory and practice has moved a bit closer together.

(a) The $a_r(p)$ and the $a_w(p)$ of the optimally mapped TLP, the mapped TLP generated by the vertex sorting approach, and the origin TLP.

(b) The $c_r(p)$ and the $c_w(p)$ of the optimally mapped TLP, the mapped TLP generated by the vertex sorting approach, and the origin TLP.

Figure 11.3: Operation available and costs measures of the vertex sorting approach.

Bibliography

[1] Robert Schadek and Oliver E. Theel. “Increasing the Accuracy of Cost and Availability Predictions of Quorum Protocols”. In: 22nd IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2017. 2017. ISBN: 978-1-5090-5652-1. doi: 10.1109/PRDC.2017.22. url: https://doi.org/10.1109/PRDC.2017.22 (cit. on pp. 1, 49, 265).

[2] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. Concurrency Control and Recovery in Database Systems. Addison Wesley, Feb. 1987, p. 370. ISBN: 13 978-0201107159 (cit. on pp. 1, 13, 15).

[3] Susan B. Davidson, Héctor García-Molina, and Dale Skeen. “Consistency in Partitioned Networks”. In: ACM Computing Surveys 17.3 (Sept. 1985), pp. 341–370. ISSN: 0360-0300 (cit. on p. 2).

[4] Chienwen Wu. “Replica Control Protocols that Guarantee High Availability and Low Access Cost”. PhD thesis. University of Illinois, Urbana-Champaign, 1993 (cit. on pp. 2, 25).

[5] Dahlia Malkhi. “Quorum Systems”. In: The Encyclopedia of Distributed Computing. Ed. by Joseph Urban and Partha Dasgupta. Kluwer Academic Publishers, 2000 (cit. on p. 2).

[6] G. Link. One Hundred Years of Russell’s Paradox: Mathematics, Logic, Philosophy. De Gruyter series in logic and its applications. Walter de Gruyter, 2004. ISBN: 9783110174380. url: https://books.google.de/books?id=Xg6QpedPpcsC (cit. on p. 7).

[7] S. Shen et al. Basic Set Theory. Student mathematical library. American Mathematical Society, 2002. ISBN: 9780821827314. url: https://books.google.de/books?id=LBvpfEMhurwC (cit. on p. 7).

[8] Helal A. Abdelsalam, Heddaya A. Abdelsalam, and Bhargava B. Bharat. Replication Techniques in Distributed Sytems. Kluwer Academic Publishers, 1996 (cit. on pp. 11, 15).

[9] Richard D. Schlichting and Fred B. Schneider. “Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems”. In: ACM Transactions on Computer Systems 1.3 (Aug. 1982), pp. 222–238 (cit. on p. 11).

[10] Keith Devlin. Fundamentals of contemporary set theory. New York: Springer-Verlag, 1979. ISBN: 0387904417 (cit. on p. 16).

[11] Robert H. Thomas. “A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases”. In: ACM Transactions on Database Systems 4.2 (June 1979), pp. 180–207 (cit. on pp. 25, 26).

[12] A. Kumar. “Performance Analysis of a Hierarchical Quorum Consensus Algorithm for Replicated Objects”. In: 10th International Conference on Distributed Computer Systems. IEEE Computer Society Press, 1990, pp. 378–385 (cit. on pp. 25, 29).

[13] Chienwen Wu and Geneva G. Belford. “The Triangular Lattice Protocol: A Highly Fault Tolerant and Highly Efficient Protocol for Replicated Data”. In: Proceedings of the 11th Symposium on Reliable Distributed Systems (SRDS’92). IEEE Computer Society Press, Oct. 1992 (cit. on pp. 25, 35).

[14] Hans-Henning Koch. “Entwurf und Bewertung von Replikationsverfahren (in German)”. PhD thesis. Department of Computer Science, University of Darmstadt, Germany, June 1994 (cit. on pp. 26, 29).

[15] Shun Yan Cheung, Mostafa H. Ammar, and Mustaque Ahamad. “The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data”. In: IEEE Transactions on Knowledge and Data Engineering 4.6 (Dec. 1990), pp. 582–592 (cit. on p. 29).

[16] Robert Schadek, Oliver Kramer, and Oliver E. Theel. “Predicting Read- and Write-Operation Availabilities of Quorum Protocols based on Graph Properties”. In: Proceedings of the 10th International Conference on Agents and Artificial Intelligence, ICAART 2018, Volume 2, Funchal, Madeira, Portugal, January 16-18, 2018. Ed. by Ana Paula Rocha and Jaap van den Herik. SciTePress, 2018, pp. 550–558. ISBN: SBN 978-989-758-275-2. doi: 10.5220/0006645705500558. url: https://doi.org/10.5220/0006645705500558 (cit. on pp. 45, 265).

[17] Reinhard Diestel. Graph Theory, 4th Edition. Vol. 173. Graduate texts in mathematics. Springer, 2012, pp. I–XVIII, 1–436. ISBN: 978-3-642-14278-9 (cit. on pp. 45, 116–118).

[18] Gary Chartrand, Linda Lesniak, and Ping Zhang. Graphs & Digraphs, Fifth Edition. 5th. Chapman & Hall/CRC, 2010. ISBN: 9781439826270 (cit. on p. 48).

[19] N. S. Altman. “An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression”. In: The American Statistician 46.3 (1992), pp. 175–185. doi: 10.1080/00031305.1992.10475879. url: https://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879 (cit. on p. 110).

[20] T. Cover and P. Hart. “Nearest neighbor pattern classification”. In: IEEE Transactions on Information Theory 13.1 (Jan. 1967), pp. 21–27. ISSN: 0018-9448. doi: 10.1109/TIT.1967.1053964 (cit. on p. 110).

[21] Joel Grus. Data science from scratch : first principles with Python. Sebastopol, CA: O’Reilly, 2015. ISBN: 978-1-491-90142-7 (cit. on p. 114).

[22] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: CoRR abs/1502.03167 (2015). url: http://arxiv.org/abs/1502.03167 (cit. on p. 114).

[23] Linton C. Freeman. “A Set of Measures of Centrality Based on Betweenness”. In: Sociometry 40.1 (1977), pp. 35–41. ISSN: 00380431. url: http://www.jstor.org/stable/3033543 (cit. on p. 118).

[24] Linton C. Freeman. “A Set of Measures of Centrality Based on Betweenness”. In: Sociometry 40.1 (1977), pp. 35–41. ISSN: 00380431. url: http://www.jstor.org/stable/3033543 (cit. on p. 119).

[25] Igor V. Tetko, David J. Livingstone, and Alexander I. Luik. “Neural network studies. 1. Comparison of overfitting and overtraining”. In: Journal of Chemical Information and Computer Sciences 35.5 (1995), pp. 826–833. doi: 10.1021/ci00027a006. url: https://pubs.acs.org/doi/abs/10.1021/ci00027a006 (cit. on p. 121).

[26] Sylvain Arlot and Alain Celisse. “A survey of cross-validation procedures for model selection”. In: Statist. Surv. 4 (2010), pp. 40–79. doi: 10.1214/09-SS054. url: https://doi.org/10.1214/09-SS054 (cit. on p. 121).

[27] Robert Schadek and Oliver Theel. “A Universal Quorum Protocol for N-Dimensional Structures”. In: Proceedings of the 1st Argentinian National Conference on Engineer Informatics and Information Systems (CoNaIISI ’13). Ed. by Marcelo M. Marciszack, Roberto M. Munoz, and Mario A. Groppo. Cordoba, Argentina: Red de Carreras de Ingenieria Informatica / Sistemas de Information (RIISIC), Nov. 2013. ISSN: 2346-9927 (cit. on pp. 131, 265).

[28] Robert Schadek and Oliver Theel. “A Graph Suite Generator for Real World Quorum Protocol Analysis”. In: Proceedings of the 2nd Argentinian National Conference on Engineer Informatics and Information Systems (CoNaIISI ’14). San Luis, Argentina: Red de Carreras de Ingeniería Informática / Sistemas de Informatión (RIISIC), Nov. 2014. ISSN: 2346-9927 (cit. on pp. 131, 165, 265).

[29] V. Feinberg, A. Levin, and E. Rabinovich. “Hypergraph Planarization”. In: VLSI Planarization: Methods, Models, Implementation. Dordrecht: Springer Netherlands, 1997, pp. 45–86. ISBN: 978-94-011-5740-7. doi: 10.1007/978-94-011-5740-7_4. url: https://doi.org/10.1007/978-94-011-5740-7_4 (cit. on p. 134).

[30] Robert Schadek and Oliver Theel. “A Data Replication Protocol for Real-World Network Topologies”. In: Proceedings of the 7th Information Technologies in Environmental Engineering International Conference. Ed. by Brenda Scholtz, Jorge Marx Gómez, and Clayton Burger. 2015, pp. 168–172. ISBN: 978-1-920508-60-9 (cit. on pp. 165, 265).

[31] Robert Schadek and Oliver E. Theel. “Crossing - A Highly Available Quorum Protocol for Arbitrary Planar Topologies”. In: Algorithms and Architectures for Parallel Processing - 15th International Conference, ICA3PP 2015, Zhangjiajie, China, November 18-20, 2015. Proceedings, Part III. Ed. by Guojun Wang et al. Vol. 9530. Lecture Notes in Computer Science. Springer, 2015, pp. 717–728. ISBN: 978-3-319-27137-8. doi: 10.1007/978-3-319-27137-8_52. url: http://dx.doi.org/10.1007/978-3-319-27137-8_52 (cit. on p. 165).

[32] Chandler Carruth. CppCon 2017: “Going Nowhere Faster” - YouTube. Nov. 2017. url: https://www.youtube.com/watch?v=2EWejmkKlxs (visited on 04/24/2018) (cit. on p. 199).

[33] Valgrind Home. url: http://valgrind.org/ (visited on 05/16/2018) (cit. on p. 199).

[34] Perf Wiki. url: https://perf.wiki.kernel.org/index.php/Main_Page (visited on 05/16/2018) (cit. on p. 199).

[35] Holger Brunst and Matthias Weber. “Custom Hot Spot Analysis of HPC Software with the Vampir Performance Tool Suite”. In: Sept. 2012, pp. 95–114. ISBN: 978-3-642-37348-0. doi: 10.1007/978-3-642-37349-7_7 (cit. on p. 200).

[36] Matt Godbolt. x86 Internals for Fun & Profit - YouTube. Aug. 2014. url: https://www.youtube.com/watch?v=hgcNM-6wr34 (visited on 04/24/2018) (cit. on pp. 204, 211, 218).

[37] Broadwell - Microarchitectures - Intel - WikiChip. url: https://en.wikichip.org/wiki/intel/microarchitectures/broadwell_(client) (visited on 04/25/2018) (cit. on p. 204).

[38] 7-cpu.com. Intel Broadwell. https://www.7-cpu.com/cpu/Broadwell.html. (Accessed on 05/10/2018) (cit. on p. 204).

[39] Broadwell Processors - HECC Knowledge Base. https://www.nas.nasa.gov/hecc/support/kb/broadwell-processors_529.html. (Accessed on 05/10/2018) (cit. on p. 205).

[40] Moritz Lipp et al. “Meltdown: Reading Kernel Memory from User Space”. In: 27th USENIX Security Symposium (USENIX Security 18). 2018 (cit. on p. 206).

[41] Paul Kocher et al. “Spectre Attacks: Exploiting Speculative Execution”. In: 40th IEEE Symposium on Security and Privacy. 2019 (cit. on p. 206).

[42] Functions - D Programming Language. (Accessed on 04/17/2018). url: https://dlang.org/spec/function.html#interpretation (cit. on p. 207).

[43] Suyog Sarda and Mayur Pandey. LLVM Essentials. Packt Publishing, 2015. ISBN: 1785280805 (cit. on p. 208).

[44] GIMPLE (GNU Compiler Collection (GCC) Internals). https://gcc.gnu.org/onlinedocs/gccint/GIMPLE.html. (Accessed on 09/19/2020) (cit. on p. 208).

[45] LLVM Language Reference Manual — LLVM 7 documentation. url: https://llvm.org/docs/LangRef.html (visited on 04/18/2018) (cit. on p. 208).

[46] LLVM's Analysis and Transform Passes — LLVM 7 documentation. url: https://llvm.org/docs/Passes.html#transform-passes (visited on 04/18/2018) (cit. on p. 208).

[47] Matt Godbolt. CppCon 2017: "What Has My Compiler Done for Me Lately? Unbolting the Compiler's Lid" - YouTube. Oct. 2017. url: https://www.youtube.com/watch?v=bSkpMdDe4g4 (visited on 04/24/2018) (cit. on p. 212).

[48] LLVM Link Time Optimization: Design and Implementation — LLVM 12 documentation. https://llvm.org/docs/LinkTimeOptimization.html. (Accessed on 09/19/2020) (cit. on p. 212).

[49] Intel. Pentium Pro Family Developer's Manual Volume 1: Specifications. Intel, 1995. url: https://downloadcenter.intel.com/de/product/49948/Intel-Pentium-Pro-Prozessor-150-MHz-256-KB-Cache-60-MHz-FSB (cit. on p. 212).

[50] Karel Driesen and Urs Hölzle. "The Direct Cost of Virtual Function Calls in C++". In: Proceedings of the 11th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. OOPSLA '96. San Jose, California, USA: ACM, 1996, pp. 306–323. ISBN: 0-89791-788-X. doi: 10.1145/236337.236369. url: http://doi.acm.org/10.1145/236337.236369 (cit. on p. 219).

[51] Robert Warshall Floyd. "Algorithm 97: Shortest Path". In: Commun. ACM 5.6 (June 1962), pp. 345–. ISSN: 0001-0782. doi: 10.1145/367766.368168. url: http://doi.acm.org/10.1145/367766.368168 (cit. on p. 223).

[52] E.W. Dijkstra. "A note on two problems in connexion with graphs". English. In: Numerische Mathematik 1.1 (1959), pp. 269–271. ISSN: 0029-599X. doi: 10.1007/BF01386390. url: http://dx.doi.org/10.1007/BF01386390 (cit. on p. 223).

[53] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: IEEE Transactions on Systems Science and Cybernetics SSC-4(2) (1968), pp. 100–107 (cit. on p. 223).

[54] Hermann Schweizer, Maciej Besta, and Torsten Hoefler. "Evaluating the Cost of Atomic Operations on Modern Architectures". In: 2015 International Conference on Parallel Architecture and Compilation, PACT 2015, San Francisco, CA, USA, October 18-21, 2015. IEEE Computer Society,

2015, pp. 445–456. ISBN: 978-1-4673-9524-3. doi: 10.1109/PACT.2015.24. url: https://doi.org/10.1109/PACT.2015.24 (cit. on p. 228).

[55] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: Commun. ACM 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782. doi: 10.1145/1327452.1327492. url: http://doi.acm.org/10.1145/1327452.1327492 (cit. on p. 228).

[56] Divyakant Agrawal and Amr El Abbadi. "The Generalized Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data". In: ACM Transactions on Database Systems 17.4 (Dec. 1992), pp. 689–717 (cit. on p. 233).

[57] David Peleg and Avishai Wool. Crumbling Walls: A Class of High Availability Quorum Systems. Technical Report CS94-07. Faculty of Mathematical Sciences, Weizmann Institute of Science, Rehovot, Israel, Jan. 1994 (cit. on p. 233).

[58] C. Berge. The Theory of Graphs. Dover books on mathematics. Dover, 2001. ISBN: 9780486419756. url: https://books.google.de/books?id=h5BjnaoKyOwC (cit. on p. 233).

[59] R. Duncan Luce and Albert D. Perry. "A method of matrix analysis of group structure". In: Psychometrika 14.2 (June 1949), pp. 95–116. ISSN: 1860-0980. doi: 10.1007/BF02289146. url: https://doi.org/10.1007/BF02289146 (cit. on p. 233).

[60] A. Cimikowski and P. Shope. "A neural-network algorithm for a graph layout problem". In: IEEE Transactions on Neural Networks 7.2 (Mar. 1996), pp. 341–345. ISSN: 1045-9227. doi: 10.1109/72.485670 (cit. on p. 235).

[61] YOSHIYASU TAKEFUJI and KUO-CHUN LEE. "A Near-Optimum Parallel Planarization Algorithm". In: Science 245.4923 (1989), pp. 1221–1223. ISSN: 0036-8075. doi: 10.1126/science.245.4923.1221. url: http://science.sciencemag.org/content/245/4923/1221 (cit. on p. 235).

[62] Robert Schadek and Oliver E. Theel. "Crossing - A Highly Available Quorum Protocol for Arbitrary Planar Topologies". In: Algorithms and Architectures for Parallel Processing - 15th International Conference, ICA3PP. 2015, pp. 717–728. ISBN: 978-3-319-27137-8. doi: 10.1007/978-3-319-27137-8_52 (cit. on p. 265).

Appendix

List of Glossary Entries and Acronyms

1SR one-copy serializability. 1, 2, 11, 13–15, 25, 29, 35, 135, 153, 167, 174

$a_r(p)$ read operation availability. 16, 17, 19, 21, 23, 26, 32, 35, 39, 45, 47, 49, 52, 55, 58–62, 64, 66, 70, 72, 76, 78, 82, 84, 88, 90, 94, 95, 98, 100, 104, 106, 110, 122, 123, 126, 128, 129, 131, 149, 150, 152, 153, 155, 157, 159, 161, 163, 165, 174–182, 186, 190, 199, 210, 235, 237, 239, 247–251, 253, 254

$a_w(p)$ write operation availability. 16, 17, 19, 21, 23, 26, 32, 35, 39, 45, 47, 49, 52, 55, 58–62, 65, 67, 71, 73, 77, 79, 83, 85, 89, 91, 96, 99, 101, 105, 107, 110, 122–124, 126, 128, 130, 131, 149, 150, 152, 153, 155, 157, 159, 161, 163, 165, 174–181, 183, 187, 191, 199, 210, 235, 237, 239, 247–251, 253, 254

AGF aggregation function. 114, 122–124, 126, 128–130, 250

ALU arithmetic-logical-unit. 217, 219

ARW average read and write availability value. 49, 51, 52, 58, 146, 153, 174, 175, 195, 230, 238, 251

asm A assemble language. 200, 212–216, 219

AVX2 advanced vector extensions. 212

BC betweenness centrality. 119–121, 123, 126, 131, 197, 233, 235, 237, 238, 253, 255

C A widely used imperativ programming language. 227

C++ A programming language, partially based on the C programming language. 199

$c_r(p)$ read operation cost. 16, 17, 20, 22, 23, 26, 32, 39, 45, 47, 55, 58–62, 68, 74, 80, 86, 92, 97, 102, 108, 110, 122–124, 128, 131, 149, 151, 153, 156, 158, 160, 162, 164, 166, 174–176, 178, 180, 181, 184, 188, 192, 199, 210, 235, 238, 239, 247, 250, 253, 254

$c_w(p)$ write operation cost. 16, 17, 20, 22, 23, 26, 32, 39, 45, 47, 55, 58–62, 69, 75, 81, 87, 93, 103, 109, 110, 122–124, 128, 131, 149, 151, 153, 156, 158, 160, 162, 164, 166, 174–176, 178, 180, 181, 185, 189, 193, 199, 210, 235, 238, 239, 247, 250, 253, 254

CFG control flow graph. 210

CIP Circle Protocol. 5, 133, 135, 136, 142, 146–167, 175, 178–180, 195, 220, 229, 233, 237, 250, 251, 259

connectivity Connectivity describes minimal number of vertices that need to be removed to partition a GS. 117, 121

CP Crossing Protocol. 5, 139, 167, 168, 170–193, 195, 220, 229, 233, 237, 251, 252, 259

CPU central processing unit. 199, 200, 204, 205, 208, 214, 219, 220, 229, 230, 234, 235

CT compile-time. 199, 209, 210, 215, 227, 257

CV cross-validation. 122, 250

D A programming language, influenced by many programming languages. 199, 200, 209–211, 213–215, 219, 222, 223, 226, 257

degree The degree describes the number of edges a vertex is connected to. 117, 118, 121, 253

distance The distance describes length of certain paths within a GS. 117–119, 121, 253

DO data object. 1–3, 11–15, 247

DR data replication. 2, 11

DRS data replication system. 11–13, 15, 25, 247

DS distributed system. 11, 13, 225–227, 229, 230, 257

DST data-structure. 220–223, 230, 257

feature A feature is a property of a class of objects that is used to compare to instances of this class. 111–114, 116–123, 125–127, 129, 130, 197, 233, 235, 237, 238, 250, 253

FI function inlining. 210, 211, 214, 257

FWA floyd-warshall path finding algorithm. 225

GP Grid Protocol. 15, 25, 29, 31–35, 39, 43, 44, 58, 60–62, 70–81, 98–103, 122–124, 126, 128–130, 153, 230, 247–249, 259

GS graph structure. 4, 8, 9, 12, 25, 29, 31, 33, 35–40, 42, 45–48, 51, 55–62, 64–110, 116–121, 123, 126, 133–139, 141–146, 149–168, 170, 174, 175, 179–193, 199, 220–223, 226, 227, 233, 235–238, 244, 245, 247–253, 255, 257

IR intermediate representation. 210, 212

***k*NN** K-nearest neighbor. 110–112, 114, 116, 122–131, 195, 197, 233, 235, 250, 253, 254

L1DC L1 data cache. 205

L1IC L1 instruction cache. 205

LNT logical network topology. 25, 45, 47–49, 52, 58, 61, 62, 122, 195, 220, 230, 233, 235–238, 248

LTO link-time-optimizations. 214

MA mapping approach. 220

MAD median absolute deviation. 59–62, 66, 67, 72, 73, 78, 79, 84, 85, 90, 91, 95, 96, 100, 101, 106, 107, 248, 249

MC middcir2. 199, 212, 220–222, 225, 227, 229, 230, 233, 235, 257

MCS Majority Consensus Protocol. 3, 4, 14, 25–28, 45, 47, 52–54, 58–62, 64–69, 94–97, 123, 124, 126, 128–130, 181, 195, 223–225, 227, 231, 233, 247–249, 252–254, 259

middcir2 middcir2. 209

MSE mean squared error. 115, 116, 122–124, 126, 128–130, 235, 253, 254

µ-op micro-operation. 214–219, 257

order Order describes how many vertices a GS consists of. 116, 117

PNT physical network topology. 3, 5, 25, 45–53, 55, 56, 58, 62, 63, 110, 116, 123, 133, 135, 136, 146–149, 153, 167, 171–173, 175, 195–197, 199, 220, 223, 229, 230, 233, 235–238, 248, 252, 253, 259

QP quorum protocol. 1–3, 5, 7, 12, 15–18, 23, 25, 29, 32, 45, 47–52, 55, 58–63, 122–124, 126, 128–130, 153, 154, 175, 195–197, 199, 209, 223, 225–227, 229–231, 233, 235, 236, 238, 248, 253, 259

ROWA Read-One/Write-All. 230

RQ read quorum. 1–3, 15–17, 25, 26, 29, 30, 32, 34, 35, 38, 39, 45, 47, 52, 61, 133, 134, 146–148, 167, 170–172, 174, 175, 225, 227, 229–231, 247, 251

RQS read quorum set. 16–20, 26, 32, 35, 49–53, 126, 195, 223–230, 233, 253, 254, 257

SD standard deviation. 59–62, 66, 67, 72, 73, 78, 79, 84, 85, 90, 91, 95, 96, 100, 101, 106, 107, 248, 249

SIMD single instruction multiple data. 207, 212, 213, 257

SQP structured quorum protocol. 25, 29, 35

SSA static single assignment. 210

TLP Triangular Lattice Protocol. 25, 35–44, 58, 60–62, 82–93, 104–109, 123, 124, 126, 128–130, 167, 175, 179, 180, 199, 229, 230, 233, 235, 237, 239, 248–251, 259

UQP unstructured quorum protocol. 25, 45

VN version number. 2, 3, 13, 14, 247

WQ write quorum. 1–3, 13, 15–17, 25, 26, 29, 30, 35, 38, 39, 45, 61, 133, 134, 146, 147, 167, 171–174, 181, 225, 231, 247, 251

WQS write quorum set. 16–20, 26, 32, 35, 49–52, 195, 223, 225, 226, 229, 230, 233, 253

List of Figures

List of Tables

List of Listings

List of Algorithms

List of Publications

Robert Schadek and Oliver Theel. “A Universal Quorum Protocol for N-Dimensional Structures”. In: Proceedings of the 1st Argentinian National Conference on Engineer Informatics and Information Systems (CoNaIISI ’13). Ed. by Marcelo M. Marciszack, Roberto M. Munoz, and Mario A. Groppo. Cordoba, Argentina: Red de Carreras de Ingenieria Informatica / Sistemas de Information (RIISIC), Nov. 2013. ISSN: 2346-9927

Robert Schadek and Oliver Theel. “A Graph Suite Generator for Real World Quorum Protocol Analysis”. In: Proceedings of the 2nd Argentinian National Conference on Engineer Informatics and Information Systems (CoNaIISI ’14). San Luis, Argentina: Red de Carreras de Ingeniería Informática / Sistemas de Informatión (RIISIC), Nov. 2014. ISSN: 2346-9927

Robert Schadek and Oliver E. Theel. “Crossing - A Highly Available Quorum Protocol for Arbitrary Planar Topologies”. In: Algorithms and Architectures for Parallel Processing - 15th International Conference, ICA3PP. 2015, pp. 717–728. ISBN: 978-3-319-27137-8. doi: 10.1007/978-3-319-27137-8_52

Robert Schadek and Oliver Theel. “A Data Replication Protocol for Real-World Network Topologies”. In: Proceedings of the 7th Information Technologies in Environmental Engineering International Conference. Ed. by Brenda Scholtz, Jorge Marx Gómez, and Clayton Burger. 2015, pp. 168–172. ISBN: 978-1-920508-60-9

Robert Schadek and Oliver E. Theel. “Increasing the Accuracy of Cost and Availability Predictions of Quorum Protocols”. In: 22nd IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2017. 2017. ISBN: 978-1-5090-5652-1. doi: 10.1109/PRDC.2017.22. url: https://doi.org/10.1109/PRDC.2017.22

Robert Schadek, Oliver Kramer, and Oliver E. Theel. “Predicting Read- and Write-Operation Availabilities of Quorum Protocols based on Graph Properties”. In: Proceedings of the 10th International Conference on Agents and Artificial Intelligence, ICAART 2018, Volume 2, Funchal, Madeira, Portugal, January 16-18, 2018. Ed. by Ana Paula Rocha and Jaap van den Herik. SciTePress, 2018, pp. 550–558. ISBN: SBN 978-989-758-275-2. doi: 10.5220/0006645705500558. url: https://doi.org/10.5220/0006645705500558

Erklärung

Hiermit erkläre ich, Robert Schadek, dass ich diese Arbeit eigenständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe. Ebenso versichere ich, dass ich diese Dissertation nur in diesem Promotionsverfahren eingereicht habe.

Robert Schadek, Oldenburg, den 2021-04-18