Computer Networks 76 (2015) 191–206

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

On the tradeoff of availability and consistency for quorum systems in data center networks

Xu Wang, Hailong Sun *, Ting Deng, Jinpeng Huai

School of Computer Science and Engineering, Beihang University, Beijing 100191, China

ARTICLE INFO

Article history:
Received 23 May 2014
Received in revised form 15 September 2014
Accepted 10 November 2014
Available online 15 November 2014

Keywords:
Quorum systems
Availability
Data center networks

ABSTRACT

Large-scale distributed storage systems often replicate data across servers and even geographically-distributed data centers for high availability, while existing theories like CAP and PACELC show that there is a tradeoff between availability and consistency. Thus eventual consistency is proposed to provide highly available storage systems. However, current practice is mainly experience-based and lacks quantitative analysis for identifying a good tradeoff between the two. In this work, we are concerned with providing a quantitative analysis on availability for widely-used quorum systems in data center networks. First, a probabilistic model is proposed to quantify availability for typical data center networks: 2-tier basic tree, 3-tier basic tree, fat tree and folded clos, and even geo-distributed data center networks. Second, we analyze replica placements on network topologies to obtain maximal availability. Third, we build the availability-consistency table and propose a set of rules to quantitatively make tradeoff between availability and consistency. Finally, with Monte Carlo based simulations, we validate our presented quantitative results and show that our approach to make tradeoff between availability and consistency is effective.

1. Introduction

Data centers as a mainstay for large-scale distributed storage systems, often face the challenges of high availability and scalability [1,2]. They typically replicate data across multiple servers and even geographically-distributed data centers [3] to tolerate network partition and node crashes. However, the CAP [4] and PACELC [5] theorems show that there is a tradeoff between availability and consistency in data replication. High availability is regarded as an important property in service level agreements for cloud services. For example, Amazon EC2 claims the availability of 99.95% [6]; Google Cloud Storage's service level commitment provides 99.9% availability [7]; and the availability of Microsoft Windows Azure is promised as 99.9% [8]. In order to provide extremely high availability, systems such as Amazon Dynamo [9] and Apache Cassandra [10] often eschew strong consistent replicas and instead provide eventual consistency [11,12]. But eventual consistency may lead to inconsistency such as read staleness [13] and write conflicts [9], since the newest value of a data item is eventually returned and the same data item may be updated on different, potentially disconnected replicas. Therefore it is necessary to understand how much availability can be obtained in different consistency levels such that we can quantitatively make tradeoff between them.

Distributed quorums [14] are widely used for eventually consistent storage systems in data centers [9,10,15,16]. With quorum replication, these storage systems write a data item by sending it to a set of replicas (write quorums) and read from a possibly different set of replicas (read quorums). Strong consistency is provided

* Corresponding author. Tel.: +86 13811909091.
E-mail addresses: wangxu@act.buaa.edu.cn (X. Wang), sunhl@act.buaa.edu.cn (H. Sun), dengting@act.buaa.edu.cn (T. Deng), huaijp@buaa.edu.cn (J. Huai).

http://dx.doi.org/10.1016/j.comnet.2014.11.006
1389-1286/

by guaranteeing the intersection of write quorums and read quorums; otherwise, read requests may return stale values, thus resulting in inconsistency. In practical quorum systems, any W replicas are defined as write quorums and any R replicas as read ones. With N replicas, if $W + R > N$, strong consistency is achieved since the overlapping of write and read quorums is always guaranteed; and if $W + R \leqslant N$, the overlapping condition may not be satisfied, therefore it can cause inconsistency. The former is a strict quorum system [17] and the latter is a probabilistic quorum system [18]. Almost all practical quorum systems like Dynamo and Cassandra provide the configuration capability on W and R. It is obvious that greater W and R will result in stronger consistency, but will lower system availability since greater W and R require more live replicas to be accessed for available write and read requests during failures. How to configure (W, R) to get the best solution for both consistency and availability? The current practice is mainly experience-based and lack quantitative analysis for identifying a good tradeoff between them.

The quantitative consistency analysis for quorum systems has been well studied [13,18]. However, availability is network topology-aware because network partition is the main cause of system unavailability according to the CAP theorem [4], and data center networks are complex [19–21] such as basic tree, fat tree, folded clos network and even geo-distributed data centers, therefore the quantitative analysis on availability of quorum systems in data center networks is challenging. There are several bodies of work [18,22–26] on quantifying the availability of quorum systems, but they have two limitations: (1) they usually assume simple network topologies such as the fully connected network and ring topology; and (2) they define the availability of quorum systems as the probability that at least one quorum is alive, but in practice live quorums may be inaccessible due to failures of switches in networks.

In this work, we focus on evaluating the availability of quorum systems in four typical data center networks: 2-tier basic tree, 3-tier basic tree, fat tree and folded clos network, and even geo-distributed data centers [19–21]. At first, we propose a system model $QS(DCN, PM, W/R)$ for quorum systems, where DCN represents the data center network topologies containing core switches, aggregation switches, ToR (Top of Rack) switches, servers and their links, PM is a vector describing the placement of replicas on servers, and W/R are the sizes of write/read quorums. Since write requests should reaches at least W live replicas and read requests must wait for responses from at least R replicas, they are equivalent to each other for our availability analysis and we only consider write requests. Based on $QS(DCN, PM, W)$, the write availability is defined as the probability that a write can reach one live replica (i.e. the coordinator) from live core switches and at least other $W - 1$ live replicas from the coordinator. Although typical data center networks are complex especially for the fat tree and folded clos network, they are essentially tree-like. Therefore, after calculating the write availability for a simple 2-tier basic tree, we use super nodes to represent subtrees or groups of nodes for 3-tier basic tree, fat tree and folded clos network which are logically equivalent to each other and obtain simplified network topologies. Then the write availability for the 3-tier basic tree, fat tree and folded clos network is computed by conditional probability and reusing the result of 2-tier basic tree. In addition, we extend our quantitative write availability from one single data center to geo-distributed data centers. We also analyze the impact of replica placement on maximizing write availability. Based on the quantitative results of availability, we build an availability-consistency table filled with values of $\langle Availability, Consistency\rangle$ for quorum systems, and propose a set of rules to choose the best (W, R) to make tradeoff between availability and consistency. Finally, through Monte Carlo based event driven simulations, we validate our quantitative results and show the effectiveness of our method for balancing availability and consistency.

We make the following contributions:

- We propose a system model $QS(DCN, PM, W/R)$ for quorum systems in typical data center networks. Unlike earlier work, our system model considers practical complex data center network DCN where switches may fail, and the placement of replicas on servers PM;
- On the basis of $QS(DCN, PM, W)$, we build a probabilistic model for the write availability of quorum systems in four typical data center networks: 2-tier basic tree, 3-tier basic tree, fat tree and folded clos network. We also extend the write availability from one single data center to geo-distributed data centers;
- We analyze how to place replicas on network topologies to maximize write availability and discuss special cases when the network topology is a 2-tier basic tree and the number of replicas is a popular value 3;
- We build an availability-consistency table filled with $\langle Availability, Consistency\rangle$, and then propose a set of rules to choose the best (W, R) configuration for a specific quorum system to balance availability and consistency;
- With Monte Carlo based event driven simulations, we validate our quantitative results and show the effectiveness of our proposed approach to quantitatively make tradeoffs between availability and consistency.

The remainder of this paper is structured as follows. Section 2 introduces the background. Section 3 presents our system model. Section 4 describes how to quantify the write availability of quorum systems. Section 5 propose the impact of replica placements on availability. How to make tradeoffs between availability and consistency is shown in Section 6. Section 7 provides our experimental results. Related work and discussion are presented in Section 8 and Section 9, respectively. Finally, Section 10 concludes the work.

2. Background

In this section, we present the background, including the preliminary knowledge of quorum systems and data center networks.

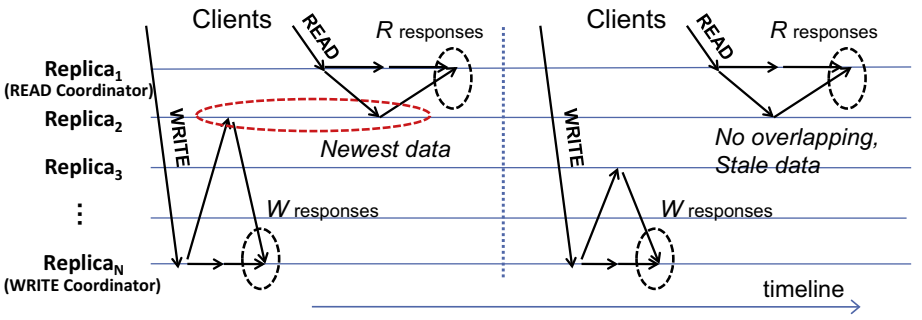


Fig. 1. An example of quorum system.

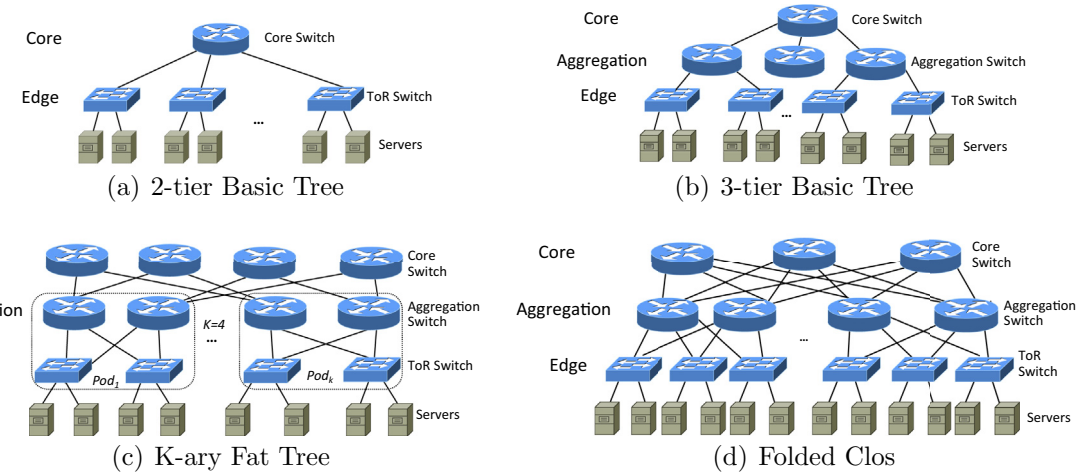


Fig. 2. Typical data center networks.

2.1. Quorum systems

Quorums have long been used in distributed storage systems [27], which is actually a set system that is a set of subsets of all replicas U. In practical quorum systems, two parameters W and R are predefined for N replicas, that is $N = |U|$ and $1 \leqslant W,\ R \leqslant N$. Let the write quorum set be $S_w = \{Q | Q \subseteq U \wedge |Q| = W\}$ and the read quorum set be $S_r = \{Q | Q \subseteq U \wedge |Q| = R\}$. Obviously, if $W + R > N$, two elements from S_w and from S_r, respectively, must intersect, thus any read can return the newest write and guarantees strong consistency. On the contrary, if $W + R \leqslant N$, the two elements overlap with a probability $P_{overlap}$:

$$P_{overlap} = 1 - \frac{\binom{N}{W}\binom{N-W}{R}}{\binom{N}{W}\binom{N}{R}} = 1 - \frac{\binom{N-W}{R}}{\binom{N}{R}} \tag{1}$$

where $\varepsilon = \binom{N-W}{R} \Big/ \binom{N}{R}$ can also be viewed as an important measurement of inconsistency for quorum systems [13].

Fig. 1 shows an example for the quorum system. For a write request, firstly it is sent to a replica as the coordinator, and then the coordinator propagates the write to all other replicas. After getting at least $W - 1$ responses, the coordinator returns. A read request is processed in the same way as the write request but the coordinator should wait for at least $R - 1$ responses. If the read set of replicas overlaps with the write one, the read request can return the newest value; otherwise, it will return stale data.

According to [22,24,25], assuming that each replica fails independently with probability p and replicas are fully connected, the availability of practical quorum systems (denoted by *Avail*) is the probability that at least one quorum survives:

$$Avail = 1 - \sum_{i=N-W+1}^{N} \binom{N}{i} p^i (1-p)^{N-i} \tag{2}$$

Since read requests perform in the same way as write requests, we only consider writes in Eq. (2). This kind of availability has two limitations: (a) replicas are often fully connected or are in other simple topologies like a ring. This assumption does not consider switch failures in practical complex networks; (b) it potentially assumes that the surviving quorums are always accessible for writes. However, as analyzed in [28], the possibility that writes can reach a replica in the surviving quorums is also an important factor for system availability. Both limitations are unified in practical network topologies where servers and switches may fail, and only part of switches can receive outward write requests. Thus it is necessary to discuss the availability of quorum systems in practical data center networks.

2.2. Data center networks

Data center networks can be classified to switch-centric topologies [19–21] and server-centric topologies [29]. Since the server-centric topologies are based on multiple-port servers, they are promising but not widely-used nowadays. Thus we only consider switch-centric topologies in this work.

Switch-centric topologies of data center networks are tree-like but differ in the tiers, the links and the number of switches. As shown in Fig. 2, they usually have three-tier switches (or routers): core tier in the root of the tree, aggregation tier in the middle and edge tier at the leaves of the tree. The core switches (CS), the aggregation switches (AS) and the ToR (Top of Rack) switches are in different tiers.

Typical tree-based data center networks include 2-tier basic tree, 3-tier basic tree, K-ary fat tree and folded clos network, and the details of these four network topologies are as follows:

1. The 2-tier basic tree does not have the aggregation tier. It includes only one core switch which connects to a number of ToR switches and underlying servers;
2. The 3-tier basic tree includes only one core switch which connects to a number of aggregation switches, and each aggregation switch links some ToR switches and corresponding servers;
3. The K-ary fat tree has K pods and $(K/2)^2$ K-port core switches. Every pod includes $K/2$ K-port aggregation switches and $K/2$ K-port ToR switches, and each aggregation switch connects to all ToR switches in the same pod. In addition, the ith core switch connects to the $([i/(K/2)]+1)$th aggregation switch in all pods; and
4. The folded clos network is built based on two parameters: D_A and D_I, where D_A denotes the double number of core switches and D_I denotes the number of aggregation switches. Thus a folded clos network has $D_A/2$ D_I-port core switches and D_I D_A-port aggregation switches, and each core switch connects to all aggregation switches. Every $D_A/2$ ToR switches connect to two same aggregation switches.

3. System model

In this section, we will propose a quorum system model that is aware of the data center network topologies and the placement of replicas.

Our system model for quorum systems (QSs) in data center networks is a triple $QS(DCN, PM, W/R)$, where

- $DCN \in$ {2-tier basic tree, 3-tier basic tree, K-ary fat tree, folded clos}; that is, DCN is one of the four typical data center networks as described above.
- PM is a 0/1 vector representing the placement of replicas on servers.
- W/R describe the sizes of write quorums and read quorums.

In the four data center networks, we assume there are three kinds of switches: core switches (10-GigE switches) with the crash probability P_c, aggregation switches (10-GigE switches) with the crash probability P_a and ToR switches (1-GigE switches) with the crash probability P_t. We also assume that the links never crash, servers are homogenous with the crash probability P_s, and the crash events are independent.

When data replicas are placed on servers, they should be put on different physical machines or virtual machines across different physical ones to improve availability. Thus "servers" in this work refer to physical machines as well as virtual machines on them, which are also the ones connecting to ToR switches in data center networks. Without loss of generality, we assume there is an order for all devices with the same type, and we use 1 and 0 to represent a replica on and not on a server, respectively. So PM is a 0/1 vector whose length equals the number of servers (denoted by n_{server}), and the number of 1 is the number of replicas (N). That is $PM = \langle r_1^s, r_2^s, \ldots, r_{n_{server}}^s \rangle$, where $r_i^s = 0$ or $1 (1 \leqslant i \leqslant n_{server})$ and $\sum_{i=1}^{n_{server}} r_i^s = N$. In addition, we define $PM^{tor} = \langle r_1^t, r_2^t, \ldots, r_{n_{tor}}^t \rangle$ where n_{tor} denotes the number of ToR switches and element r_i^t means the number of replicas that the ith ToRS can access from downlinks, thus we have $\sum_{i=1}^{n_{tor}} r_i^t = N$. Similarly, PM^{as} for aggregation switches can be defined. For example, there are 3 replicas (i.e., $N = 3$) for a 3-tier basic tree in Fig. 3, then $PM = \langle 0,1,1,0,0,1,0,0 \rangle$, $PM^{tor} = \langle 1,1,1,0 \rangle$ and $PM^{as} = \langle 2,1 \rangle$.

In practical quorum systems, W and R are the number of replicas for write and read, respectively. An available read performs similarly with an available write, thus read availability can be obtained by replacing W with R in the results of write availability, and we only consider write availability in the next section.

4. Quantitative analysis of availability

Given the replication placement vector PM and the value of W, the write availability of a quorum system in one data center network is defined as the probability that a write request reaches at least one live replica through core switches and the replica can arrive at least other $(W-1)$ live replicas. In this section, we will compute the write availability for four typical data center networks: 2-tier basic tree (bt2), 3-tier basic tree (bt3), k-ary fat tree (ft) and folded clos network (fc). In addition, the write availability is extended from one single data center to geo-distributed data centers.

4.1. Sketch of quantitative analysis

At first, although practical data center networks usually have thousands of and even tens of thousands of nodes, our analysis focuses on the replicas of one data item and the involved sub-network since the number of replicas is usually not big (e.g., Cassandra has a default configuration of $N = 3,\ W = R = 1$ [30]). As shown in Fig. 4, three replicas (i.e., $N = 3$) of a data item are placed on a 3-tier basic tree which may have thousands of nodes, but we only need to analyze the sub-network marked by the dash line since the involved sub-network includes all possible routing paths from the core switch to N replicas. The servers and

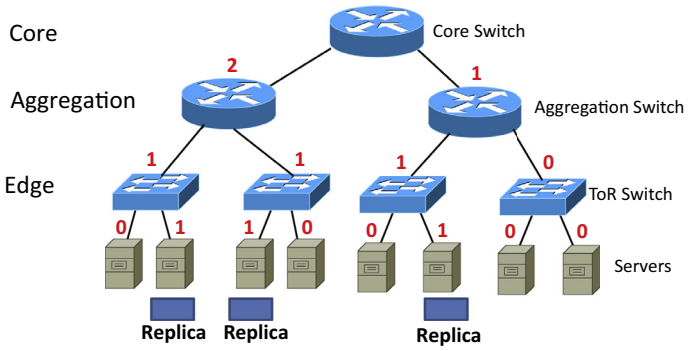


Fig. 3. Replica placement.

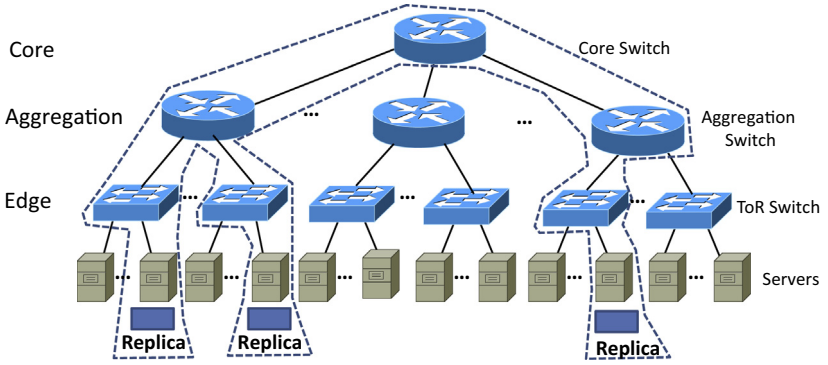


Fig. 4. Involved sub-network.

switches outside the sub-network will not influence our results and can be ignored. In addition, the replica placement *PM* is a variable in our analysis, thus the sub-network should cover all possible placements of N replicas. As a result, there are usually N servers or switches connecting to each switch of the upper layer in the involved sub-network in case that we put N replicas underlying a same ToR switches or distribute N replicas among N different servers connecting to different ToR switches and even different aggregation switches.

Next we will show our quantitative analysis sketch of the involved sub-networks for the four typical data center networks. Because it is complex to directly compute the write availability especially for the fat tree and folded clos network, we begin from the simplest 2-tier basic tree and then extend the 2-tier basic tree step by step to 3-tier basic tree, fat tree and folded clos. As depicted in Fig. 5, when the write availability of 2-tier basic tree is calculated, we add the aggregation tier to the 2-tier basic tree and then get 3-tier basic tree. By reducing 2-tier sub-trees of 3-tier basic tree to super nodes, an equivalent 2-tier basic tree is obtained and thus we can reuse the result of 2-tier basic tree. Then we further add K pods and $(K/2)^2$ core switches to get the K-ary fat tree, and add $D_A/2$ core switches and D_I aggregation switches to get the folded clos network, respectively. For the fat tree and the folded clos network, we first use super nodes to represent groups of core switches and aggregation switches which are logically equivalent to each other and then get simplified network topologies which are (or similar with) 3 tier basic trees,

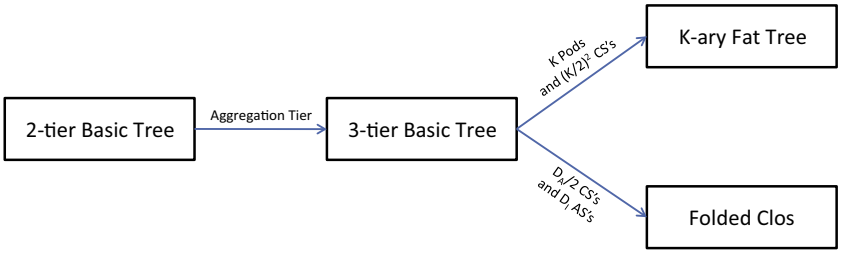


Fig. 5. Sketch of quantitative analysis.

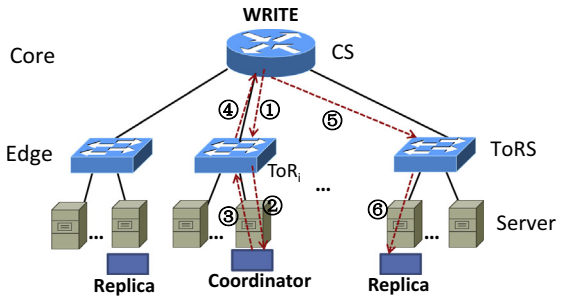


Fig. 6. Available write for 2-tier basic tree.

thus we can simplify the computation of the fat tree and the folded clos network by reusing the result of 3-tier basic tree.

4.2. 2-Tier basic tree

Typically, a two-tier basic tree network can support 5–8 K servers [20] in data centers. Since only the core switch can receive write requests, we just consider the involved sub-network which includes all possible routing paths from the core switch to N replicas. The sub-network contains N ToRSs, and each ToRS connects to N servers, thus any placement of N replicas is possible. As shown in Fig. 6, if a write request reaches the CS, it will be sent to one live replica as the coordinator via a live ToRS, and then the coordinator sends the write request to other replicas via ToRSs and the CS. After getting $(W-1)$ acknowledgement (ACK) messages, the coordinator will return the write and thus the write is available. Otherwise, the write is unavailable.

It is obvious that if a write is available in a two-tier basic tree network, the write can access at least W live replicas via the live CS; and inversely, if a write can access at least W live replicas via the live CS, the write must be available. Therefore the event that a write is available is equivalent to the event that a write can access at least W live replicas via the live CS.

Let stochastic variable A_i be the number of accessible live replicas underlying the ith ToRS from the live CS, where $1 \leqslant i \leqslant N$. Given $PM = \langle r_1^s, r_2^s, \ldots, r_{n_{server}}^s \rangle (n_{server} = N^2)$, we can get $PM^{tor} = \langle r_1^t, r_2^t, \ldots, r_N^t \rangle$ where $r_i^t = \sum_{j=(i-1)N+1}^{iN} r_j^s$. Then we calculate the probability density function (pdf) of A_i as follows:

$$Pr(A_i = m_i) = \begin{cases} P_t + (1-P_t)(P_s)^{r_i^t} & r_i^t \geqslant m_i = 0; \\ (1-P_t)\binom{r_i^t}{m_i}(1-P_s)^{m_i}(P_s)^{r_i^t - m_i} & r_i^t \geqslant m_i > 0. \end{cases}$$

where $m_i = 0$ if the ith ToRS fails, or the ith ToRS is alive and all r_i^t servers with replicas fail; and $m_i > 0$ if the ith ToRS is alive and exact m_i servers with replicas are alive. Recall that P_t and P_s are the crash probability of ToRS and servers, respectively, and have been defined in Section 3.

Let stochastic variable $X_{bt2-tor}$ be the number of accessible live replicas from the live CS in the two-tier basic tree network, and the probability of $X_{bt2-tor} = x$ is the sum of the probability for all possible cases that $A_1 + A_2 + \cdots + A_N = x$. Since the CS is alive and PM^{tor} is given, $A_1, \ldots, A_N$ are independent. Thus we have:

$$Pr(X_{bt2-tor} = x) = \sum_{m_1+m_2+\cdots+m_N=x} \prod_{i=1}^{N} Pr(A_i = m_i)$$

where $0 \leqslant m_i \leqslant r_i^t$. Define a function $Q_{bt2-tor}(x) = Pr(X_{bt2-tor} = x)$ for simplicity, then the probability that a write can access at least W live replicas via the live CS is $(1-P_c)\sum_{x=W}^{N} Q_{bt2-tor}(x)$. That is:

$$Avail(QS(bt2, PM, W)) = (1-P_c)\sum_{x=W}^{N} Q_{bt2-tor}(x) \tag{3}$$

where $Avail(QS(bt2, PM, W))$ denotes the write availability of quorum systems in 2-tier basic tree where the replica placement is PM and the size of write quorums is W. Similarly, $Avail(QS(bt3, PM, W))$ for 3-tier basic tree, $Avail(QS(ft, PM, W))$ for fat tree and $Avail(QS(fc, PM, W))$ for folded clos can be defined.

4.3. 3-Tier basic tree

In the three-tier basic tree, we also only consider the possible involved sub-network. The sub-network has N ASs, and each AS connects to N ToRSs. Any ToRS also connects to N servers. Fig. 7 shows an example. If a write request reaches the CS, it will be sent to one live replica which is the coordinator via a live AS and a live ToRS, and then the coordinator sends the write to other replicas through ToRSs, ASs and the CS. When the coordinator receives $(W-1)$ ACK messages from other replicas, it will return the write result and then the write is available. Otherwise, the write is unavailable. Similarly, we can see that, the event that a write is available is equivalent to the event that a write can access at least W live replicas via the live CS in the 3-tier basic tree.

Let stochastic variable B_j be the number of accessible live replicas under the jth AS from the live CS, where $1 \leqslant j \leqslant N$. Given $PM = \langle r_1^s, r_2^s, \ldots, r_{n_{N^3}}^s \rangle$, we can get $PM^{tor} = \langle r_1^t, r_2^t, \ldots, r_{N^2}^t \rangle$ where $r_i^t = \sum_{j=(i-1)N+1}^{iN} r_j^s$, and $PM^{as} = \langle r_1^a, r_2^a, \ldots, r_N^a \rangle$ where $r_i^a = \sum_{j=(i-1)N+1}^{iN} r_j^t$. As shown in Fig. 7, the sub-trees marked by dash lines can be seen as two-tier basic trees when we replace ASs with CSs, thus we calculate the pdf of B_j by reusing the results of two-tier basic trees:

$$Pr(B_j = m_j') = \begin{cases} P_a + (1-P_a)Q_{bt2-tor}(0) & r_i^a \geqslant m_j' = 0; \\ (1-P_a)Q_{bt2-tor}(m_j') & r_i^a \geqslant m_j' > 0. \end{cases}$$

where $m_j' = 0$ if the jth AS fails, or the jth AS is alive and the sub-tree whose root is the jth AS can access 0 live replicas; and $m_j' > 0$ if the jth AS is alive and there are exact m_j' replicas which can be accessed in the sub-tree whose root is the jth AS.

Let stochastic variable X_{bt3-as} be the number of accessible live replicas from the live CS in the three-tier basic tree network, and the probability of $X_{bt3-as} = x'$ is the sum of the probability for all possible cases that $B_1 + B_2 + \cdots +$

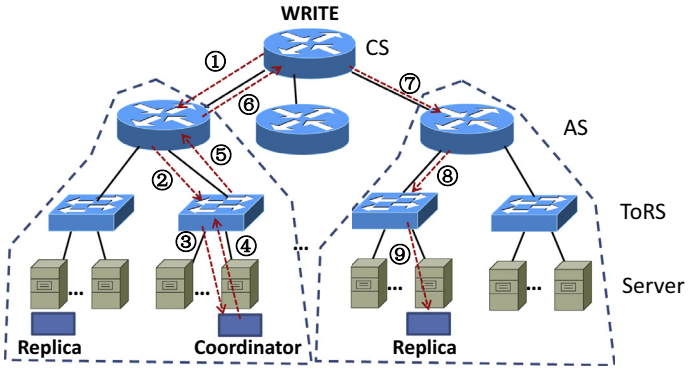


Fig. 7. Available write for 3-tier basic tree.

$B_N = x'$. Since the CS is alive and PM^{as} is given, $B_1, \ldots, B_N$ are independent. Thus:

$$Pr(X_{bt3-as} = x') = \sum_{m'_1+m'_2+\cdots+m'_N=x'} \prod_{j=1}^{N} Pr(B_j = m'_j)$$

where $0 \leqslant m'_j \leqslant r^a_j$.

Because the write availability of three-tier basic tree equals the probability that a write can access at least W live replicas via the live CS, we have:

$$Avail(QS(bt3, PM, W)) = (1 - P_c) \sum_{x'=W}^{N} Pr(X_{bt3-as} = x') \qquad (4)$$

4.4. Fat tree

The K-ary fat tree has K pods and $(K/2)^2$ CSs. There are $K/2$ ASs and $K/2$ ToRSs in a pod where each AS connects to all ToRSs, and each ToRS connects to N servers. In addition, the ith CS connects to the $([i/(K/2)]+1)$th aggregation switch in all pods. From Fig. 8 we can see that, if we put every $(K/2)$ CSs into a group from left to right, we can have $(K/2)$ groups of CSs and the jth group of CSs are all connecting to the jth AS in every pods. As shown in Fig. 8, a write request can be sent to any live CS and reaches one live replica which is the coordinator via a live AS and a live ToRS in one pod, and then the coordinator sends the write to other replicas through ToRSs, ASs and one CS. If $(W-1)$ ACK messages from other replicas are received by the coordinator, the write will be returned to clients and it is available.

To simplify the K-ary fat tree, we use a core switch group (GCS) to represent a group of CSs since they connect to the same ASs, and use a super aggregation switches (SAS) to represent all ASs in a pod because they connect to the same ToRSs. For a GCS, its crash probability P_{gc} is the probability that all internal CSs crash, thus $P_{gc} = (P_c)^{K/2}$. If there are exact $x (1 \leqslant x \leqslant K/2)$ live GCSs, the crash probability P_{sa} for a SAS is the probability that all the internal x ASs which connect to the x live GCSs crash, thus $P_{sa} = (P_a)^x$. Fig. 9 shows GCSs and SASs in the simplified fat tree.

Let stochastic variable C_j be the number of accessible live replicas underlying the jth SAS from the x live GCSs, where $1 \leqslant j \leqslant K$. Given $PM = \langle r^s_1, r^s_2, \ldots, r^s_{n_{server}} \rangle$, we can get $PM^{tor} = \langle r^t_1, r^t_2, \ldots, r^t_{K^2/2} \rangle$ where $r^t_i = \sum_{j=(i-1)N+1}^{iN} r^s_j$, and $PM^{sas} = \langle r^{sa}_1, r^{sa}_2, \ldots, r^{sa}_K \rangle$ where $r^{sa}_i = \sum_{j=(i-1)K/2+1}^{iK/2} r^t_j$. As shown in Fig. 9, pods can be seen as two-tier basic trees when we replace SASs with CSs, thus we can compute the pdf of C_j when there are exact x live GCSs:

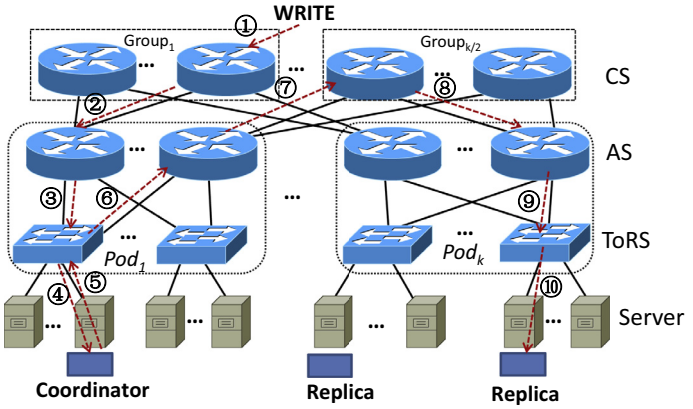


Fig. 8. Available write for K-ary fat tree.

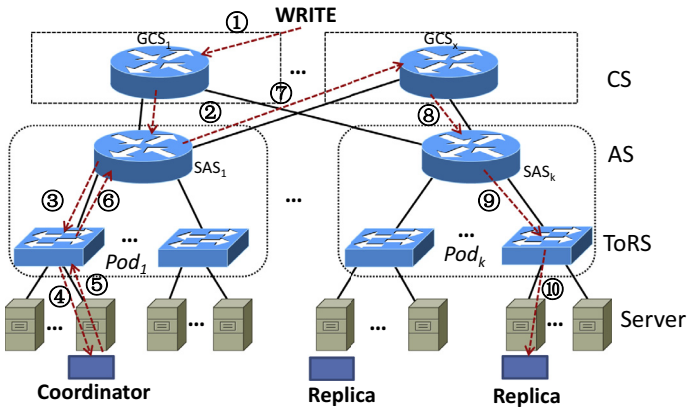


Fig. 9. GCSs and SASs in fat tree.

$$Pr(C_j = m'_j) = \begin{cases} P_{sa} + (1 - P_{sa})Q_{bt2-tor}(0) & r_i^{sa} \geqslant m'_j = 0; \\ (1 - P_{sa})Q_{bt2-tor}(m'_j) & r_i^{sa} \geqslant m'_j > 0. \end{cases}$$

where $m'_j = 0$ if the jth SAS fails, or the jth SAS is alive and the sub-tree whose root is the jth SAS can access 0 live replicas; and $m'_j > 0$ if the jth SAS is alive and there are exact m'_j replicas which can be accessed in the sub-tree whose root is the jth SAS.

Let stochastic variable X_{ft-sas} be the number of accessible live replicas in the fat tree network and the number of live GCSs is denoted by stochastic variable X_{gcs}. When there are exact x live GCSs, the probability of $X_{ft-sas} = x'$ is the sum of the probability for all possible cases that $C_1 + C_2 + \cdots + C_K = x'$. Since every live GCS connects to all K SASs, $C_1, \ldots, C_K$ are independent. Thus:

$$Pr(X_{ft-sas} = x' | X_{gcs} = x) = \sum_{m'_1 + m'_2 + \cdots + m'_K = x'} \prod_{j=1}^{K} Pr(C_j = m'_j)$$

where $0 \leqslant m'_j \leqslant r_j^{sa}$. At the same time, we can have:

$$Pr(X_{gcs} = x) = \binom{K/2}{x} (1 - P_{gc})^x (P_{gc})^{K/2-x}$$

Then we can get $Pr(X_{ft-sas} = x')$ by summing the conditional probability for each possible $x (1 \leqslant x \leqslant K/2)$:

$$Pr(X_{ft-sas} = x') = \sum_{x=1}^{K/2} Pr(X_{ft-sas} = x' | X_{gcs} = x) Pr(X_{gcs} = x)$$

Finally the probability that a write can access at least W live replicas in the K-ary fat tree is $\sum_{x'=W}^{N} Pr(X_{ft-sas} = x')$, which can also be seen as the probability that a write is available. That is:

$$Avail(QS(ft, PM, W)) = \sum_{x'=W}^{N} Pr(X_{ft-sas} = x') \quad (5)$$

Note that, if a write is available, it must be able to access at least W live replicas in the K-ary fat tree. However, when a write can access at least W live replicas in the fat tree where the W live replicas can be accessed only by CSs in different groups, the write coordinator may not cross multiple CSs to reach other replicas, but this possibility is very small since partitioning two CSs in different groups requires cutting so many redundant paths between them in the fat tree. Therefore Eq. (5) is a conservative upper bound on write availability of K-ary fat tree but accurate enough for us.

4.5. Folded clos network

The folded clos network includes $D_A/2$ CSs and D_I ASs. Any CS and any AS are connected, and every $D_A/2$ ToRSs connect to two ASs. And each ToRS connects to N servers. As shown in Fig. 10, a write request can be sent to any live CS and reaches one live replica as the coordinator via a live AS and a live ToRS, and then the coordinator sends the write to other replicas through ToRSs, ASs and one CS. If $(W - 1)$ ACK messages from other replicas are received by the coordinator, the write will reply and makes itself available. Similarly, the event that a write is available is equivalent to the event that a write can access at least W live replicas in the folded clos network since CSs and ASs are fully connected.

Like the K-ary fat tree, we use a GCS to represent all CSs since they connect to the same ASs, and use a SAS to represent two ASs which connect to the same ToRSs. Fig. 11 shows the GCS and SASs in the simplified folded clos. For the GCS, its crash probability P'_{gc} is the probability that all CSs crash, thus $P'_{gc} = (P_c)^{D_A/2}$; For the SASs, the crash probability P'_{sa} for a SAS is the probability that both internal ASs crash, thus $P'_{sa} = (P_a)^2$.

Let stochastic variable D_j be the number of accessible live replicas underlying the jth SAS from the GCS, where $1 \leqslant j \leqslant D_I/2$. Given $PM = \langle r_1^s, r_2^s, \ldots, r_{n_{server}}^s \rangle$, we have $PM^{tor} = \langle r_1^t, r_2^t, \ldots, r_{D_A D_I/4}^t \rangle$ where $r_i^t = \sum_{j=(i-1)N+1}^{iN} r_j^s$, and $PM^{sas} = \langle r_1^{sa}, r_2^{sa}, \ldots, r_{D_I/2}^{sa} \rangle$ where $r_i^{sa} = \sum_{j=(i-1)D_A/2+1}^{iD_A/2} r_j^t$. As shown in Fig. 11, the sub-trees marked by dash lines can be seen as two-tier basic trees when we replace SASs with CSs, thus we calculate the pdf of D_j by reusing the results of two-tier basic trees:

$$Pr(D_j = m'_j) = \begin{cases} P'_{sa} + (1 - P'_{sa})Q_{bt2-tor}(0) & r_i^{sa} \geqslant m'_j = 0; \\ (1 - P'_{sa})Q_{bt2-tor}(m'_j) & r_i^{sa} \geqslant m'_j > 0. \end{cases}$$

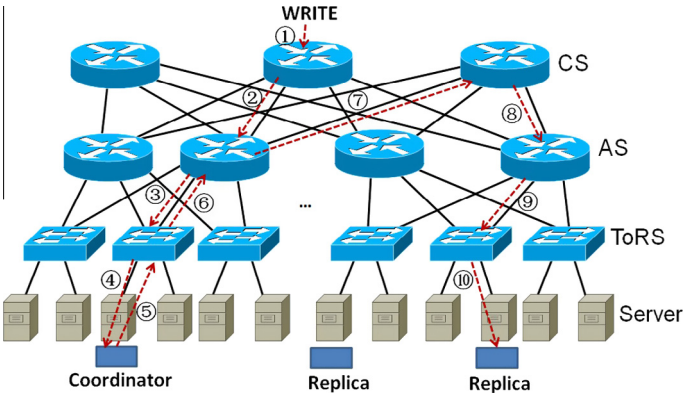


Fig. 10. Available write for folded clos network.

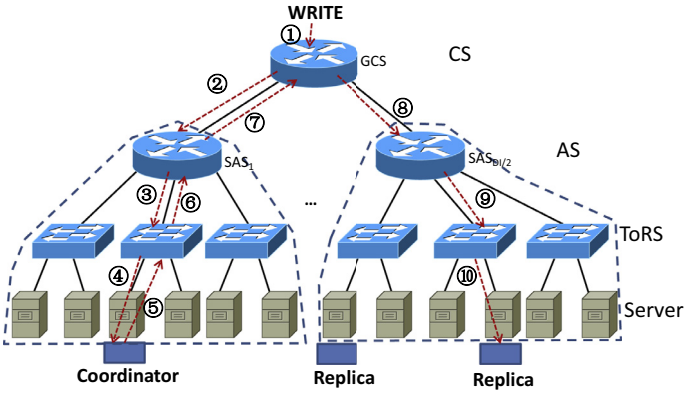


Fig. 11. GCSs and SASs in folded clos.

where $m'_j = 0$ if the jth SAS fails, or the jth SAS is alive and the sub-tree whose root is the jth SAS can access 0 live replicas; and $m'_j > 0$ if the jth SAS is alive and there are exact m'_j replicas which can be accessed in the sub-tree whose root is the jth SAS.

Let stochastic variable X_{fc-sas} be the number of accessible live replicas from the live GCS in the folded clos network, and the probability of $X_{fc-sas} = x'$ is the sum of the probability for all possible cases that $D_1 + D_2 + \cdots + D_{D_l/2} = x'$. Since the GCS is alive, $D_1, \ldots, D_{D_l/2}$ are independent. Thus:

$$Pr(X_{fc-sas} = x') = \sum_{m'_1+m'_2+\cdots+m'_{D_l/2}=x'} \prod_{j=1}^{D_l/2} Pr(D_j = m'_j)$$

where $0 \leqslant m'_j \leqslant r^{sa}_j$. Finally we can have:

$$Avail(QS(fc, PM, W)) = (1 - P'_{gc}) \sum_{x'=W}^{N} Pr(X_{fc-sas} = x') \qquad (6)$$

4.6. Geo-distributed data centers

In above analysis, we propose the quantitative results of write availability for four kinds of data center networks. In fact, replicas are usually deployed not on one single data center but across multiple geo-distributed data centers [3] for further improving availability.

Assume that there are M geo-distributed data centers $GDCS = \{DCN_1, DCN_2, \ldots, DCN_M\}$, and we place N replicas across them by $PM = \{PM_1, PM_2, \ldots, PM_M\}$. The number of replicas in DCN_i is denoted by $|PM_i| (1 \leqslant i \leqslant M)$. Let stochastic variable X_{gdcs} be the number of accessible live replicas in the geo-distributed data centers, and X_{dcn_i} be the number of accessible live replicas in the $DCN_i (1 \leqslant i \leqslant M)$. Then the probability of $X_{gdcs} = x$ is the sum of the probability for all possible cases that $X_{dcn_1} + X_{dcn_2} + \cdots + X_{dcn_M} = x$. By Eqs. (3)–(6), we have:

$$Pr(X_{dcn_i} = x_i) = \begin{cases} (1 - P_c) Q_{bt2-tor}(x_i) & DCN_i = bt2; \\ (1 - P_c) Pr(X_{bt3-as} = x_i) & DCN_i = bt3; \\ Pr(X_{ft-sas} = x_i) & DCN_i = ft; \\ (1 - P'_{gc}) Pr(X_{fc-sas} = x_i) & DCN_i = fc. \end{cases}$$

Since these data centers are usually in different geographical locations, we assume that they are independent. Thus we have:

$$Pr(X_{gdcs} = x) = \sum_{x_1+x_2+\cdots+x_M=x} \prod_{i=1}^{M} Pr(X_{dcn_i} = x_i)$$

where $0 \leqslant x_i \leqslant |PM_i|$. As a result, we have the probability that a write can access at least W live replicas, which can also be seen as the write availability of geo-distributed data centers:

$$Avail(QS(GDCS, PM, W)) = \sum_{x=W}^{N} Pr(X_{gdcs} = x)$$

Now the write availability of quorum systems for typical data center networks can be obtained by Eqs. (3)–(6) as well as the geo-distributed data centers. Compared with Eq. (2) which is the result based on previous studies such as [22,24,25], our results are more accurate to estimate system availability of practical quorum systems in data center networks since switches can fail and our system model is more practical. In fact, when $P_c = 0,\ P_a = 0$ and $P_t = 0$, that is, all replicas are always connected and live quorums are always accessible, Eqs. (3)–(6) will be degraded to Eq. (2). In this way, previous studies can be viewed as special cases of our work on quantifying availability of practical quorum systems.

Note that since read requests are performed similarly with write requests in quorum systems, thus the read availability of quorum systems can be represented by $Avail(QS(DCN, PM, R))$ and $Avail(QS(GDCS, PM, R))$ where we replace W with R in the equations of write availability.

5. Replica placement

In this section, we will discuss the impact of replica placements on the write availability of quorum systems.

During the placements of replicas, what we want is to find the best solution to distribute replicas in one data center network topology such that we can have the maximal write availability. In fact, given the data center networks DCN (including crash probabilities of devices) and the values of $N,\ W$, the problem is how to find the best PM (PM^{tor}) for maximizing the write availability of quorum systems. It can be represented as a combinatorial optimization problem:

$$\text{Maximize } Avail(QS(DCN, PM^{tor} =< r_1^t, r_2^t, \ldots, r_{n_{tor}}^t >, W))$$

subject to

$$\begin{cases} \sum_{i=1}^{n_{tor}} r_i^t = N \\ 0 \leqslant r_i^t, r_i^t \in \mathbb{Z} (1 \leqslant i \leqslant n_{tor}) \end{cases}$$

Solving this combinatorial optimization problem is very difficult since Eqs. (3)–(6) are complicated. However, if N is small (e.g., $N < 10$), we can use computers to calculate the numerical values of Eqs. (3)–(6) and exhaustively search all feasible solutions to find the maximal value of the objective function within a few seconds. In addition, when the data center network is not very large (support less than 8 K servers [20]) which can use a 2-tier basic tree topology and the number of replicas is the common value of 3, more insightful conclusions can be provided.

For a data center network with the 2-tier basic tree topology, we have three different placements of replicas when $N = 3 : PM_1^{tor} = \langle 1,1,1 \rangle,\ PM_2^{tor} = \langle 2,1,0 \rangle,\ PM_3^{tor} = \langle 3,0,0 \rangle$. It is obvious that other replica placements are equivalent to one of the three placements. Table 1 indicates the write availability of these three different replica placements for 2-tier basic tree when $W = 1,2,3$, respectively, where $F_1 = (1-P_c)(1-P_t)(1-P_s),\ F_2 = (1-P_c)(1-P_t)(1-P_s)^2,\ F_3 = (1-P_c)(1-P_t)(1-P_s)^3,\ \bar{P}_t = 1-P_t$ and $\bar{P}_s = 1-P_s$.

Since $0 < P_c, P_t, P_s, \bar{P}_t, F_1, F_2, F_3 < 1$, thus $\bar{P}_t{}^2 F_3 < \bar{P}_t F_3 < F_3$ and we can have:

$$Avail(PM_1^{tor}, 3) < Avail(PM_2^{tor}, 3) < Avail(PM_3^{tor}, 3)$$

If $W = 2$, it is obvious that $Avail(PM_2^{tor}, 2) < Avail(PM_3^{tor}, 2)$. At the same time, we have:

$$Avail(PM_1^{tor}, 2) - Avail(PM_2^{tor}, 2) = P_t F_2 (1 - 2P_t - 2P_s + 2P_s P_t)$$

so $Avail(PM_1^{tor}, 2) > Avail(PM_2^{tor}, 2)$ if $1 - 2P_t - 2P_s + 2P_s P_t > 0$, otherwise $Avail(PM_1^{tor}, 2) \leqslant Avail(PM_2^{tor}, 2)$. Similarly, we can get:

$$Avail(PM_1^{tor}, 2) - Avail(PM_3^{tor}, 2) = P_t F_2 (1 - 4P_s - 2P_t + 2P_s P_t)$$

then if $1 - 4P_s - 2P_t + 2P_s P_t > 0$ we have $Avail(PM_1^{tor}, 2) > Avail(PM_3^{tor}, 2)$, otherwise $Avail(PM_1^{tor}, 2) \leqslant Avail(PM_3^{tor}, 2)$.

When $W = 1$, because $0 < P_c, P_t, P_s < 1$ we have:

$$Avail(PM_1^{tor}, 1) - Avail(PM_2^{tor}, 1) = P_t F_1 (P_s(1-P_s) + P_t(1-P_s)^2) > 0$$

and:

$$Avail(PM_2^{tor}, 1) - Avail(PM_3^{tor}, 1) = P_t F_1 (1 - P_s^2) > 0$$

therefore we get:

$$Avail(PM_1^{tor}, 1) > Avail(PM_2^{tor}, 1) > Avail(PM_3^{tor}, 1)$$

In summary, we show our conclusions in Table 2. We can see that the placement $PM_1^{tor} = \langle 1,1,1 \rangle$ which puts three replicas in different racks has the maximal availability when $W = 1$, while it obtains the minimal availability when $W = 3$. And the placement $PM_3^{tor} = \langle 3,0,0 \rangle$ which puts three replicas in one same rack has the opposite effect on availability. When $W = 2$, whether PM_3^{tor} or PM_1^{tor} have the maximal availability is determined by the values of P_t and P_s. In addition, we find that the placement

Table 1
Write availability for 2-tier basic tree when $N = 3$.

$Avail$	$PM_1^{tor} = \langle 1,1,1 \rangle$	$PM_2^{tor} = \langle 2,1,0 \rangle$	$PM_3^{tor} = \langle 3,0,0 \rangle$
$W = 1$	$(3 - 3\bar{P}_t\bar{P}_s + \bar{P}_t{}^2\bar{P}_s{}^2)F_1$	$(1 + P_s + P_t + P_s^2\bar{P}_t)F_1$	$(1 + P_s + P_s^2)F_1$
$W = 2$	$\bar{P}_t(1 + 2P_t + 2P_s\bar{P}_t)F_2$	$(1 + 2P_s\bar{P}_t)F_2$	$(1 + 2P_s)F_2$
$W = 3$	$\bar{P}_t{}^2 F_3$	$\bar{P}_t F_3$	F_3

Table 2
Availability comparison for 2-tier basic tree with different replica placements.

W	Availability comparison	
1	$Avail(PM_1^{tor}) > Avail(PM_2^{tor}) > Avail(PM_3^{tor})$	
2	$Avail(PM_1^{tor}) \leqslant Avail(PM_2^{tor}) < Avail(PM_3^{tor})$	$1 + 2P_sP_t \leqslant 2P_s + 2P_t$
	$Avail(PM_2^{tor}) < Avail(PM_1^{tor}) \leqslant Avail(PM_3^{tor})$	$2P_s + 2P_t < 1 + 2P_sP_t \leqslant 4P_s + 2P_t$
	$Avail(PM_2^{tor}) < Avail(PM_3^{tor}) < Avail(PM_1^{tor})$	$1 + 2P_sP_t > 4P_s + 2P_t$
3	$Avail(PM_1^{tor}) < Avail(PM_2^{tor}) < Avail(PM_3^{tor})$	

$PM_2^{tor} = \langle 2, 1, 0\rangle$ which places two replicas in one rack and the third in another (remote) rack never gets the maximal availability, and may has the worst availability if $W = 2$ and $1 + 2P_sP_t > 2P_s + 2P_t$.

In practice, the common Hadoop configuration [31] is PM_2^{tor} (two replicas in one rack and the third in another rack). Since network bandwidth between servers in the same rack is greater than network bandwidth between servers in different racks, designers say that this configuration may have a better bandwidth than PM_1^{tor} as well as a higher availability than PM_3^{tor}. However, according to Table 2, it is true only if $W = 1$, and if $W = 2$ or 3 we show that PM_3^{tor} may be better than PM_2^{tor} in both aspects of network bandwidth and availability. Ref. [32] indicates that Apache Cassandra has two different replica strategies: one is to put replicas in different nodes without considering racks, and the other is to place replicas in distinct racks. That is, the former may be one of PM_1^{tor}, PM_2^{tor} and PM_3^{tor} and its availability can be seen as the average availability of PM_1^{tor}, PM_2^{tor} and PM_3^{tor}, which is a moderate and conservative choice for availability; and the latter is our replica placement PM_1^{tor}, which is the best configuration when $W = 1$ or $W = 2$ as well as $1 + 2P_sP_t > 4P_s + 2P_t$ but may be the worst such as $W = 3$.

6. Quantitative configuration for availability and consistency

Theories like CAP and PACELC show that there is a tradeoff between availability and consistency for replication, and the current practice is more or less experience-based and lacks quantitative analysis for identifying a good tradeoff between the two. In this section, we will discuss how to make tradeoff between them for quorum systems in a quantitative way.

For a specific quorum system and all possible placements of replicas, we assume that the ratio of write requests is $\alpha(0 \leqslant \alpha \leq 1)$, thus the ratio of read requests is $1 - \alpha$. In this way, the system availability with the configuration (W, R) is:

$$Avail(W, R) = \max_{PM}(\alpha Avail(QS(W)) + (1 - \alpha)Avail(QS(R)))$$

In Section 2 we mention that $\varepsilon = \binom{N-W}{R} / \binom{N}{R}$ are often viewed as an important measurement of inconsistency for practical quorum systems [13]. In this work we also use it to measure the inconsistency, that is:

$$Consistency(W, R) = 1 - \frac{\binom{N-W}{R}}{\binom{N}{R}}$$

where $W + R \leqslant N$. When $W + R > N$, quorum systems are strongly consistent ($Consistency = 1$). Note that failures of switches and servers also affect the consistency of quorum systems, but this impact may be hidden in the probability tail and thus can be ignored [13].

In practical systems, system providers should promise a bounded availability in their service agreement levels, which is usually described by the number of nines:

$$N_n = -\lg(1 - Avail(W, R))$$

However, availability bounded by N_n (often one positive integer), it does not mean that greater N_n is better since it will cause weaker consistency of data replicas. In fact, we want to know a configuration (W, R) for a specific quorum system whose availability is no lower than the promised N_n, as well as the possible strongest consistency. Since $W, R(1 \leqslant W, R \leqslant N)$ are positive integers and N is not very big, we suggest an approach based on the availability-consistency table to look up the best configuration.

The **availability-consistency table** (ACT) consists of N rows and N columns, where the row represents the value of W and the column represents the value of R. Element $ACT(W, R)$ in the table is filled with a tuple $\langle [N_n], Consistency\rangle$. Note that N_n is rounded down to match the bounded availability. Since availability decreases and consistency increases when W and R rise, the number of nines will decline and consistency will increase from $ACT(1, 1)$ to $ACT(N, N)$. An example of the availability-consistency table is shown in Table 4.

When the bounded availability is given as one positive integer N_{n_0}, we look up all feasible elements whose availability is no lower than N_{n_0} in the availability-consistency table. But how to choose the best one when there are more than one feasible element? a set of rules is provided as follows:

1. If we want to obtain a stronger consistency at the same time, the one with smaller $[N_n]$ in two feasible elements is better;
2. If two feasible elements have the same $[N_n]$, the one with stronger consistency is better; and
3. If two feasible elements have the same $[N_n]$ and the same consistency, the one with smaller value of $\alpha W + (1 - \alpha)R$ is better since $\alpha W + (1 - \alpha)R$ can be

viewed as an estimation of system performance. For example, for a read-dominated quorum system, which means that α is near 0, we prefer that R be as small as possible if we have the same $\lceil N_n \rceil$ and the same consistency.

In fact, above rules show that we prefer to configure (W, R) by an *availability* $\rightarrow$ *consistency* $\rightarrow$*performance* priority order. But this is just an example, system providers can adapt it based on the availability-consistency table by their personalized preferences.

7. Experiments

According to Eqs. (3)–(6) as discussed in Section 4, the write availability of quorum systems depends on the types of data center networks, the failure rates of servers, ToRSs, ASs and CSs, the placement of replicas on servers, and the value of W for write quorums. In this section, we focus on validating our presented results, and discussing the trends while choosing different experimental parameters.

7.1. Monte Carlo simulation

Considering the complexity of Eqs. (3)–(6) and verifiability for experimental results, we implement $QS(DCN, PM, W)$ using Monte Carlo based event driven simulations. The data center network topologies are simulated by extending CloudSim [33]. In the simulations, each write request is randomly sent to one replica across data center networks. If any switch or server crashes, it then does not route write requests. For every ten million writes, we detect write availability by checking whether a write is answered or lead to a timeout. The number of available writes is recorded as N_{aw}. Thus we estimate $Avail(QS(DCN, PM, W))$ by $N_{aw}/10,000,000$.

7.2. Validating write availability

In this experiment, we compare our prediction results calculated by Eqs. (3)–(6) with the observed experimental values to validate them. We have tried to choose the more reasonable and practical parameters. According to the statistics of [2,34], failure rate P_s is set to 2%, 3% and 4%, P_t is 1%, 2%, 3%, 4% and 5%, P_a is one of 1%, 5% and 10%, and P_c is 1%, 2%. The placement of replicas is represented by PM^{tor}, and some replica placements may be equivalent to each other for our experiments thus we choose one of them. Obviously the number of possible placements is less than $N!$ for 2-tier basic tree and $(N!)^2$ for 3-tier basic tree, fat tree and folded clos network, which makes our simulations feasible when N is often small, e.g., Cassandra has a default configuration ($N = 3,\ W = R = 1$ [30]).

While building a 2-tier basic tree, the number of ToRS in the involved sub-network is set to N so that each replica can be connected to one ToRS, and one ToRS connects to N servers so that all N replicas can be located on servers under one same ToRS, which ensures that any placements can be possible. We repeat experiments for $N \in [3, 9],\ W \in [1, N]$ and all possible placements, and then record the value of N_{aw} to estimate $Avail(QS(bt2, PM, W))$. Comparing observed values with predicted ones of write availability in all cases, we have an average RMSE(root mean square error) = 0.000472% and std.dev. (standard deviation) = 0.000417%.

In the 3-tier basic tree, the respective sub-network has N ASs, N^2 ToRSs and N^3 servers for all possible placements. And we repeat experiments for $N \in [3, 9]$ and $W \in [1, N]$. By comparing observed values with predicted write availability in all cases, we get an average RMSE = 0.000763% and std.dev. = 0.000668%.

For the K-ary fat tree, $K \geqslant 2N$ since every ToRS connects to $K/2$ servers and we want to observe the case that all N replicas are connected to one same ToRS. Without loss of generality, we set $K = 2N$ and repeat experiments for $N \in [3, 9]$ and $W \in [1, N]$, and the average RMSE = 0.00117% and std.dev. = 0.000934% between the observed values and predicted values in all cases.

In the folded clos network, similarly $D_A \geqslant 2N$ and $D_I \geqslant 2N$ for all possible placements of replicas. Without loss of generality, we set $D_A = D_I = 2N$ and make experiments for $N \in [3, 9]$ and $W \in [1, N]$, and we find our average RMSE = 0.000845% and std. dev. = 0.000689% between the observed values and predicted write availability for all possible placements of replicas.

Table 3 shows all the experimental results of average RMSE and std. dev. between the observed values and predicted write availability for four typical data center networks, which validate our predicted results. In addition, the accuracy of our prediction is high enough to at least 4 nines of availability.

7.3. Impact of W on write availability

Eq. (3)–(6) reveals the relation of write availability and W. In this experiment, we want to show the change of write availability with respect to W more intuitively. Without loss of generality, we set $P_c = 1\%,\ P_a = 5\%,\ P_t = 2\%$ and $P_s = 2\%$. The number of replicas N is a common value 3, and the placement of replicas is the one which maximizes write availability. To highlight the variance of different write availability, we use the number of nines to represent it. As shown in Fig. 12, we can see that the number of nines decreases as W is increased for any data center networks. In addition, we can obtain different nines of write availability by tuning the value of W for the fat tree and folded clos network but may not for the 2-tier and 3-tier basic tree.

7.4. Impact of N on write availability

This experiment is to show the trend of write availability with respect to N. Likewise, $P_c = 1\%,\ P_a = 5\%,\ P_t = 2\%$ and $P_s = 2\%$, and the replica placement chooses the one which maximizes write availability. Since the fat tree and folded clos network are more widely used in practical distributed systems, we take the fat tree as an example and our experiments show that other three topologies have the same result. Fig. 13 indicates the impact of N on the nines of write availability in the K-ary fat tree ($K = 2N$) from $N = 3$ to $N = 7$. It shows that the write availability

Table 3
Write availability validation.

	RMSE (%)	Std. dev. (%)
2-Tier basic tree	0.000472	0.000417
3-Tier basic tree	0.000763	0.000668
K-Ary fat tree	0.00117	0.000934
Folded clos	0.000845	0.000689

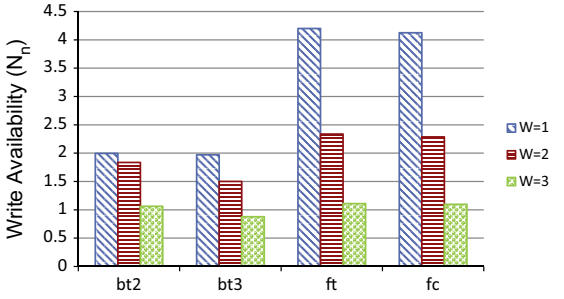


Fig. 12. Impact of W on write availability.

increases from $N = 3$ to $N = 7$ for any fixed W, and rises in steps that are at least one nine.

7.5. Impact of PM on write availability

As described in Section 5, the placement of replicas on the data center network topologies has an impact on write availability. Let us take the folded clos network ($D_A = D_I = 2N$) as an example, where the number of replicas $N = 3$ and other parameters are the same with above experiments. The placement of replicas is represented by PM^{tor} which has six different values: $PM_1^{tor} = \langle 1,0,0,1,0,0,1,0,0\rangle$, $PM_2^{tor} = \langle 2,0,0,1,0,0,0,0,0\rangle$, $PM_3^{tor} = \langle 1,1,0,1,0,0,0,0,0\rangle$, $PM_4^{tor} = \langle 2,1,0,0,0,0,0,0,0\rangle$, $PM_5^{tor} = \langle 1,1,1,0,0,0,0,0,0\rangle$ and $PM_6^{tor} = \langle 3,0,0,0,0,0,0,0,0\rangle$. That is, PM_1^{tor} evenly distributes replicas among the data center network, PM_2^{tor} and PM_4^{tor} (two in a rack and the third in anther one) are the common Hadoop configuration [31] and PM_6^{tor} concentrates replicas on one ToRS. Fig. 14 shows that the highest write availability occurs when the placement is PM_1^{tor} or PM_6^{tor}, and Hadoop placements are moderate ones between them due to the consideration of performance.

7.6. An example of availability and consistency tradeoff

This experiment is designed to show an example of quantitative (W,R) configuration for making tradeoff between availability and consistency.

Take the fat tree as an example. For a quorum system in a K-ary fat tree network ($K = 2N$), the quorum system is read dominated ($\alpha = 0.05$). The number of replicas N is 3. Based on the results of above experiments, the availability-consistency table is built in Table 4, where $P_c = 1\%$, $P_a = 5\%$, $P_t = 2\%$ and $P_s = 2\%$. When we need 3-nines availability, the best configuration of (W,R) is $(2,1)$ not $(1,1)$ if we want to obtain a stronger consistency at the same time. Similarly, $(W,R) = (1,2)$ is not a good configuration while requiring 2-nines availability. And $(W,R) = (3,1)$ is the best among $(2,2)$, $(3,1)$, $(3,2)$ because it guarantees the strong consistency as well as better performance than other two ones.

8. Related work

8.1. Quorum systems

For strict quorum systems, many authors have measured their availability and found optimal available quorum systems. Ref. [25] found that the most available strict quorum systems are non-dominated coteries. If each replica of a quorum system fails independently with probability p and replicas are fully connected, the most available quorum system is the majority system for $0 < p < 1/2$ and the singleton system for $1/2 < p < 1$.

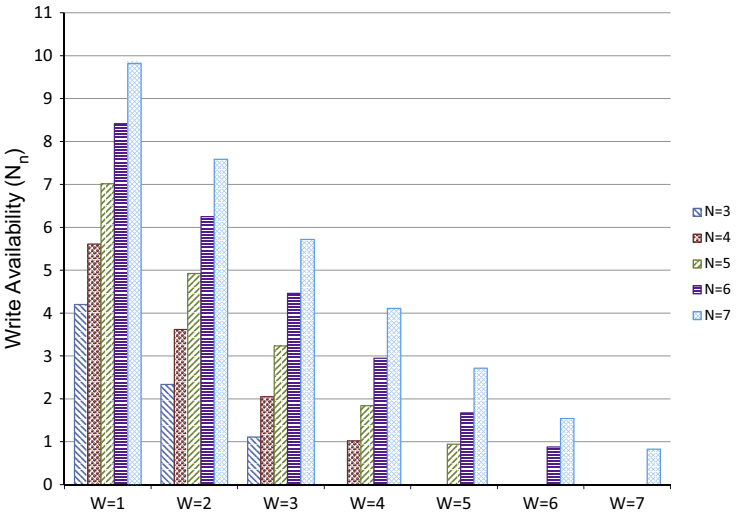


Fig. 13. Impact of N on write availability.

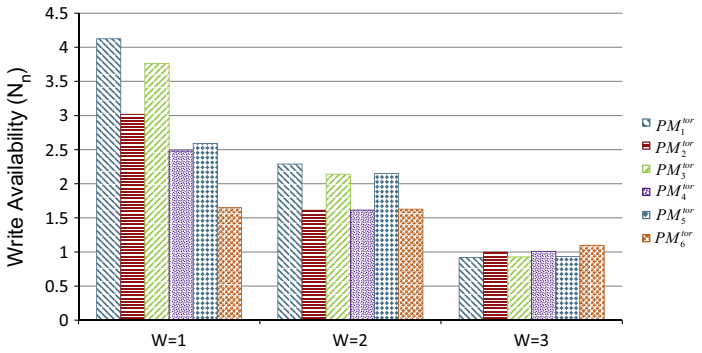


Fig. 14. Impact of *PM* on write availability.

Table 4
Availability-consistency table.

$\langle A, C\rangle$	$R = 1$	$R = 2$	$R = 3$
$W = 1$	$\langle 4, 0.333\rangle$	$\langle 2, 0.667\rangle$	$\langle 1, 1.0\rangle$
$W = 2$	$\langle 3, 0.667\rangle$	$\langle 2, 1.0\rangle$	$\langle 1, 1.0\rangle$
$W = 3$	$\langle 2, 1.0\rangle$	$\langle 2, 1.0\rangle$	$\langle 1, 1.0\rangle$

Ref. [24] discussed the tradeoffs between the availability, the load and capacity of strict quorum systems. Ref. [26] provided explicit optimal available solution for the ring topology. Ref. [23] proposed a heuristic method to compute the availability of majority systems for more general network topologies. However, our work focuses on practical (W, R) quorum systems which are usually not strict quorum systems.

For probabilistic quorum systems, Refs. [18,22] discussed their availability. Ref. [18] showed that probabilistic quorum systems can offer high availability and heavy load at the same time. Ref. [22] increased system availability by guaranteeing an upper bound on the staleness of data for probabilistic quorum systems. Both of them quantified the availability of probabilistic quorum systems based on fully-connected networks and thus did not consider the impact of network topologies on the availability. Ref. [35] quantified the availability of quorum systems in data center networks but did not analyze the impact of replica placements.

Refs. [36,28] analyzed the availability of quorum systems based on long-time empirical workload and fault observation, and found that network partition really existed and caused system unavailability. But they required much time to repeat recording historical logs for new different networks.

Practical quorum systems [9,10] provided configurable (W, R) to obtain different consistency levels, as well as different availability. But this is just a qualitative method, and our work further measures how availability can be acquired by tuning the values of (W, R).

8.2. Network reliability and failures

Ref. [37] discussed how to compute the reliability of complex networks, which showed that computing the probability that a subset of nodes remains connected in a complex network is an NP-hard problem. We take four typical data center networks as special cases and show the feasibility when the number of involving nodes (N) is not very big.

There are many studies on network failures such as [2,34,38,39]. Ref. [38] characterized failures in an ISP backbone, which observed failures as either router related or optical related by correlating time and impacted network components. Ref. [39] studied network failures based on system logs, email notifications and router configurations. Ref. [2] reported various failures in Google data centers and [34] proposed an analysis of failures in a data center network, including the reliability of network devices and links, the impact of failures on network traffic. This body of work is very useful to help us understanding network failures and reliability, but quorum systems as a widely-used replication strategy have not been discussed.

9. Discussion

In Section 3, we assume that servers and the same types of switches are homogeneous, and their crash failures are independent. In practice, data center networks may be more complex, which are built on (1) heterogeneous servers and switches, and (2) the crash events may be correlated with each other [40]. To solve the heterogeneity of servers and switches, we can use different values of P_c, P_a, P_t and P_s for different devices when Eqs. (3)–(6) are calculated. But for the correlation problem of crash events, the correlation patterns and correlation factors may depend on the implementation details of specific data center networks such as workloads, power devices and cooling systems, thus we will discuss its impact on system availability in future.

10. Conclusion

In this paper, based on our presented system model $QS(DCN, PM, W/R)$, we quantify the availability of quorum systems for typical data center networks: 2-tier basic tree, 3-tier basic tree, fat tree, folded clos network, and even for geo-distributed data centers. Then we analyze the impact of replica placements on availability and discuss special cases when the network topology is 2-tier basic tree and

the number of replicas is a popular value 3. With the quantitative results of availability, we build the availability-consistency table and propose a set of rules to choose the best (W, R) configuration for quantitatively making trade-off between availability and consistency for quorum systems. Through Monte Carlo simulations, we validate our quantitative results and show the effectiveness of our approach to quantitatively balance availability and consistency.

Acknowledgments

This work was supported by National Natural Science Foundation of China (No. 61370057), China 863 program (No. 2012AA011203), China 973 program (No. 2014CB340300), A Foundation for the Author of National Excellent Doctoral Dissertation of PR China (No. 201159), and Beijing Nova Program (2011022).

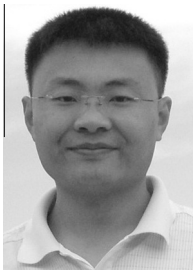
Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.comnet.2014.11.006.

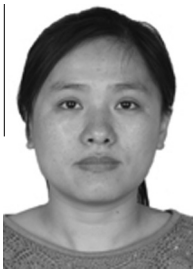
References

[1] Amazon.com, Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region, April 2014. <http://aws.amazon.com/cn/message/65648/>.
[2] J. Dean, Designs, lessons, and advice from building large distributed systems, in: LADIS Keynote, 2009.
[3] Y. Sovran, R. Power, M.K. Aguilera, J. Li, Transactional storage for geo-replicated systems, in: SOSP, 2011.
[4] E.A. Brewer, Towards robust distributed systems, in: PODC, 2000, pp. 7–7.
[5] D.J. Abadi, Consistency tradeoffs in modern distributed database system design: CAP is only part of the story, IEEE Comput. 45 (2) (2012) 37–42.
[6] Amazon EC2, Amazon EC2 Service Level Agreement, April 2014. <http://aws.amazon.com/ec2-sla/>.
[7] Google, Google Cloud Storage SLA, December 2013. <https://developers.google.com/storage/sla>.
[8] Windows Azure, Windows Azure Service Level Agreement, December 2013. <http://www.windowsazure.com/en-us/support/legal/sla/>.
[9] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: amazons highly available key-value store, in: SOSP, 2007.
[10] The Apache Cassandra Project, April 2014. <http://cassandra.apache.org/>.
[11] Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, Understanding Eventual Consistency, Technical Report MSR-TR-2013-39,Microsoft Research.
[12] W. Vogels, Eventually consistent, Commun. ACM 52 (2009) 40–44.
[13] P. Bailis, S. Venkataraman, M.J. Franklin, J.M. Hellerstein, I. Stoica, Probabilistically bounded staleness for practical partial quorums, in: VLDB, 2012.
[14] M. Merideth, M. Reiter, Selected results from the latest decade of quorum systems research, Replicat., Lect. Notes Comput. Sci. 5959 (2010) 185–206.
[15] Basho riak, April 2014. <http://basho.com/products/riak-overview/>.
[16] A. Feinberg, Project voldemort: reliable distributed storage, in: ICDE, 2011.
[17] R. Jimenez-Peris, M. Patino Martinez, G. Alonso, K. Bettina, Are quorums an alternative for data replication?, ACM Trans Database Syst. 28 (2003) 257–294.
[18] D. Malkhi, M. Reiter, A. Wool, R. Wright, Probabilistic quorum systems, Inf. Commun. 170 (2001) 184–206.
[19] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, VL2: a scalable and flexible data center network, in: SIGCOMM, 2009.
[20] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, in: SIGCOMM, 2008.
[21] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, Portland: a scalable fault-tolerant layer 2 data center network fabric, in: SIGCOMM, 2009.
[22] A. Aiyer, L. Alvisi, R.A. Bazzi, On the availability of non-strict quorum systems, in: DISC, 2005.
[23] D. Barbara, H. Garcia-Molina, The reliability of voting mechanisms, IEEE Trans. Comput. 100 (10) (1987) 1197–1208.
[24] Moni Naor, Avishai Wool, The load, capacity, and availability of quorum systems, SIAM J. Comput. 27 (2) (1998) 423–447.
[25] D. Peleg, A. Wool, The availability of quorum systems, Inf. Comput. 123 (2) (1995) 210–223.
[26] T. Ibaraki, H. Nagamochi, T. Kameda, Optimal coteries for rings and related networks, Distrib. Comput. 8 (1995) 191–201.
[27] D.K. Gifford, Weighted voting for replicated data, in: SOSP, 1979.
[28] Yu Haifeng, Amin Vahdat, The costs and limits of availability for replicated services, ACM Trans. Comput. Syst. 24 (1) (2006) 70–113.
[29] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, BCube: a high performance, server-centric network architecture for modular data centers, ACM SIGCOMM Comput. Commun. Rev. 39 (4) (2009) 63–74.
[30] Apache Cassandra, Cassandra 1.0 Thrift Configuration, April 2014. <https://github.com/apache/cassandra/blob/cassandra-1.0/interface/cassandra.thrift>.
[31] Apache Hadoop, HDFS Architecture Guide, April 2014. <http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html>.
[32] Apache Cassandra, Data Replication, December 2013. <http://www.datastax.com/documentation/cassandra/2.0/cassandra/architecture/architectureDataDistributeReplication_c.html>.
[33] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A. De Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw.: Pract. Exper. 41 (1) (2011) 23–50.
[34] P. Gill, N. Jain, N. Nagappan, Understanding network failures in data centers: measurement, analysis, and implications, ACM SIGCOMM Computer Communication Review, vol. 41, ACM, 2011, pp. 350–361.
[35] X. Wang, H. Sun, T. Deng, J. Huai, A quantitative analysis of quorum system availability in data centers, in: 2014 22nd IEEE/ACM International Symposium on Quality of Service (IWQoS), IEEE/ACM, 2014.
[36] Y. Amir, A. Wool, Evaluating quorum systems over the internet, in: Proceedings of the Annual International Symposium on Fault-Tolerant Computing, 1996.
[37] A. Rosenthal, Computing the reliability of a complex network, SIAM J. Appl. Math. 32 (1977) 384–393.
[38] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali, C. Diot, Characterization of failures in an operational ip backbone network, IEEE/ACM Trans. Netw. (TON) 16 (4) (2008) 749–762.
[39] D. Turner, K. Levchenko, A.C. Snoeren, S. Savage, California fault lines: understanding the causes and impact of network failures, ACM SIGCOMM Comput. Commun. Rev. 40 (4) (2010) 315–326.
[40] Y. Liu, J. Muppala, Fault-tolerance characteristics of data center network topologies using fault regions, in: 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, 2013, pp. 1–6.

Xu Wang received the BS degree from Beihang University, China, in 2008. Currently, he is working toward the PhD degree in the School of Computer Science and Engineering, Beihang University, Beijing, China. His research interests include the areas of distributed systems, service computing, replication, and availability.



Hailong Sun received the BS degree in computer science from Beijing Jiaotong University in 2001. He received the PhD degree in computer software and theory from Beihang University in 2008. He is an Associate Professor in the School of Computer Science and Engineering, Beihang University, Beijing, China. His research interests include services computing, cloud computing and distributed systems. He is a member of the IEEE and the ACM.



Ting Deng received the PhD degree in Computer Science from the Beihang University, Beijing China, in 2011. Currently, she is an assistant professor at Beihang University. Her research interests include distributed system, Service oriented computing and recommendation systems.

Jinpeng Huai is a Professor and President of Beihang University. He used to serve on the Steering Committee for Advanced Computing Technology Subject for the National High-Tech Program (863) as Chief Scientist. He is a member of the Consulting Committee of the Central Government's Information Office, and Chairman of the Expert Committee in both the National e-Government Engineering Taskforce and the National e-Government Standard office. His research interests include networked software, cloud computing and trustworthiness & security.