Scalable Byzantine Fault Tolerance on Heterogeneous Servers

Michael Eischer and Tobias Distler
Friedrich-Alexander University Erlangen-Nürnberg (FAU)
Email: {eischer,distler}@cs.fau.de

***Abstract*—When provided with more powerful or extra hardware, state-of-the-art Byzantine fault-tolerant (BFT) agreement protocols are unable to effectively exploit the additional computing resources: On the one hand, in settings with heterogeneous servers existing protocols cannot fully utilize servers with higher performance capabilities. On the other hand, using more servers than the minimum number of replicas required for Byzantine fault tolerance in general does not lead to improved throughput and latency, but instead actually degrades performance.**

In this paper, we address these problems with OMADA, a BFT protocol that is able to benefit from additional hardware resources. To achieve this property while still providing strong consistency, OMADA first parallelizes agreement into multiple groups and then executes the requests handled by different groups in a deterministic order. By varying the number of requests to be ordered between groups as well as the number of groups that a replica participates in between servers, OMADA offers the possibility to individually adjust the resource usage per server. Moreover, the fact that not all replicas need to take part in every group enables the protocol to exploit additional servers.

***Keywords*—Byzantine Fault Tolerance, State-Machine Replication, Scalability, Heterogeneity, Resource Efficiency**

I. INTRODUCTION

Applying the concept of Byzantine fault-tolerant (BFT) state-machine replication [1], it is possible to build reliable systems that continue to correctly provide their services even if some of their replicas fail in arbitrary ways. This includes failure scenarios caused by hardware problems as well as (potentially malicious) misbehavior of software components. In order to guarantee consistency in such systems, replicas execute client requests only after the requests have been committed by a BFT agreement protocol, which is responsible for establishing a global total order on all requests. In particular, the agreement protocol ensures that the determined order of requests remains stable in the presence of replica failures.

In general, BFT agreement protocols require a minimum of $3f+1$ replicas in order to tolerate up to f faulty replicas [2]. Although the number of participants in a BFT agreement protocol can be larger than $3f+1$, many BFT systems opt for exactly this many replicas [1], [3], [4], [5], [6], [7], [8]. This is mainly due the fact that the internal architectures of most state-of-the-art BFT agreement protocols do not allow them to exploit additional replicas. In contrast, with all replicas in the system participating in the ordering of all client requests, additional replicas usually come at the cost of an increased computational and network overhead, and consequently degrade performance without offering any notable advantages.

To prevent the agreement protocol from becoming the bottleneck of the entire BFT system, research efforts in recent years aimed at increasing the throughput of BFT agreement while keeping the number of replicas at a minimum [7], [8]. However, these approaches are based on the assumption that all replicas run on homogeneous servers, that is, servers with equal or at least similar performance capabilities. Unfortunately, it is not always possible to operate a BFT system under such conditions. Especially in cloud deployments, the performance capabilities of different virtual machines can vary significantly even if they are of the same instance type [9]. This is usually a consequence of virtual machines in the cloud data center being run on heterogeneous physical servers, making it very difficult for cloud providers to offer identical computing resources across virtual machines. As a result, it is basically impossible for a cloud user to ensure the homogeneity of virtualized servers when deploying a BFT system in the cloud.

To address the problems discussed above, in this paper we present OMADA, a protocol that enables BFT systems to exploit computing resources existing protocols are not able to utilize: additional agreement replicas as well as spare capacities on fast servers. OMADA achieves this by parallelizing agreement into multiple heterogeneous groups and varying the agreement workload between them. This approach allows OMADA to individually adjust the ordering responsibilities of replicas to the particular performance capabilities of their servers. For example, a replica on a more powerful server can participate in more than one group and/or groups that are responsible for ordering a large fraction of requests, whereas a replica on a less powerful server might only be part of a single group handling a small portion of the workload.

Although in this paper we primarily focus on heterogeneity introduced by the servers hosting the replicas of a BFT system, we expect our approach to also be beneficial for scenarios in which variations between replicas are the result of other sources of heterogeneity. For example, using heterogeneous replica implementations in order to minimize the probability of common mode failures [3], [10], [11] in general also causes replicas to advance at different speeds, consequently having a similar effect as servers with different capabilities.

In summary, this paper makes three contributions: (1) It presents OMADA, a BFT protocol that is able to benefit from additional agreement replicas. (2) It details how OMADA can exploit heterogeneous servers with different performance capabilities. (3) It evaluates OMADA in a heterogeneous setting. In the remainder of this paper, Section II identifies limitations of existing protocols, Section III presents our approach to address these issues with OMADA, Section IV evaluates OMADA, Section V discusses related work, and Section VI concludes.

This is the authors' version of an article published in the *Proceedings of the 13th European Dependable Computing Conference (EDCC '17)*, Geneva, Switzerland, 4–8 September 2017.
The final version is available from IEEE Xplore http://ieeexplore.ieee.org/document/8123550.

II. Background and Problem Statement

Below, we give an overview of BFT systems and analyze the scalability and resource usage of state-of-the-art protocols.

A. Background

In general, BFT systems based on state-machine replication [1], [3], [4], [5], [6], [7], [8] require $n \geq 3f+1$ replicas to tolerate up to f faulty replicas. As shown in Figure 1, replicas ensure consistency by first running a protocol to agree on a client request before executing the request. For this purpose, one of the replicas acts as *leader* while all others participate as *followers*. If the leader becomes faulty, replicas initiate a *view change* to reassign the leader role to a different replica.

Having received a request from a client, the leader assigns a unique sequence number to the request and then starts the agreement process consisting of two rounds of all-to-all communication between replicas: In the first round, which consists of two phases (i.e., *pre-prepare* and *prepare*), replicas ensure that they consider the same request proposal by the leader. After that, the second round (i.e., the *commit* phase) is responsible for finalizing the assignment of the sequence number to the particular request. In both cases, a replica completes a round once it has collected a quorum of size $\lceil \frac{n+f+1}{2} \rceil$ of matching messages. The quorum size guarantees that each possible pair of quorums intersects in at least $f+1$ arbitrary replicas, and therefore in at least one correct replica.

Requests for which the agreement process has completed on a replica are executed in the order of their sequence numbers. A client accepts the result to its request after having obtained $f+1$ matching replies from different replicas as this guarantees that at least one of the replies was sent by a correct replica.

In order to prevent a faulty replica from impersonating a correct replica, correct replicas authenticate each message, usually using a MAC authenticator, that is, a vector of message authentication codes [1]. Each MAC in the vector is calculated using a secret only known to the sender and a particular receiver and cannot be verified by a third party, thus requiring an authenticator to contain an individual MAC for each intended recipient of the message. As a result, both the size of a MAC authenticator and the computational cost of creating it are proportional to the number of message recipients.

B. Problem Statement

Building on the basic approach presented in Section II-A, in recent years different works have proposed architectural changes and protocol refinements, for example, to improve resilience [5] or reduce replication costs [12], [13], [14]. Below, we focus on two problems that so far remain unsolved: The ability of a BFT system to scale with the number of agreement replicas as well as the efficient use of heterogeneous servers.

Lack of Scalability. For applications for which the computational cost of executing a client request is comparably small (e.g., coordination services [15]), the agreement stage of a BFT system usually becomes the decisive factor limiting performance. Unfortunately, introducing additional agreement replicas to solve this issue is not an option in existing BFT systems. This has mainly two reasons: First, due to the fact that as discussed in Section II-A the quorum size depends on the total number of replicas, adding replicas leads to larger quorums and consequently requires more messages. Second, when the number of intended recipients increases, creating MAC authenticators for the messages exchanged between replicas becomes more costly and the messages become larger.

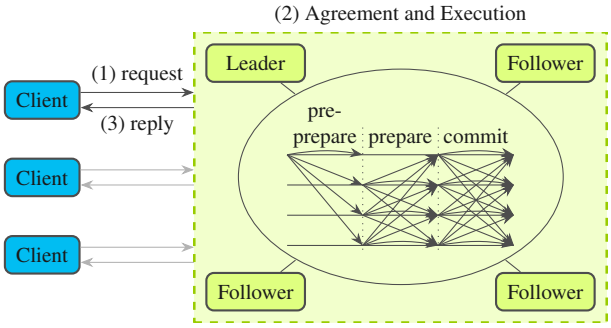


Figure 1. Basic BFT system architecture relying on state-machine replication.

Inefficient Use of Heterogeneous Servers. With all replicas participating in both the agreement and the execution of all client requests, the replicas in a BFT system usually consume a similar amount of processing resources. Some protocols even deliberately minimize potential imbalances caused by the additional responsibilities of a leader by rotating the leader role among replicas [4], [7]. While a balanced resource usage is beneficial if replicas run on servers that have the same performance capabilities, it prevents existing BFT systems from fully utilizing the available resources if replicas are executed on heterogeneous servers. Due to progress depending on a quorum of replicas, in such environments the performance of the agreement stage is limited by the $\lceil \frac{n+f+1}{2} \rceil$th fastest server, leaving resources on more powerful machines unused.

Summary. In order to be able to benefit from additional agreement replicas, a scalable BFT protocol must ensure consistency without involving all replicas in all message exchanges. Furthermore, to exploit heterogeneous servers such a protocol must provide means to distribute load depending on the specific performance capabilities of each server.

III. Omada

In this section, we present details of Omada, focusing on how the BFT protocol is able to exploit additional replicas as well as spare capacities on heterogeneous servers.

A. Architecture

As illustrated in Figure 2, to use additional servers Omada parallelizes agreement into multiple groups, each consisting of $3f+1$ agreement replicas. For the same reason, Omada in addition also separates agreement from execution [12] relying on $2f+1$ execution replicas; that is, requests not necessarily need to be processed by the same replicas by which they have been ordered. As a result, replicas in Omada may assume three different roles that are associated with different responsibilities: coordinating an agreement group (*leader*), assisting in a group (*follower*), and executing requests (*executor*).

To support heterogeneous servers, a replica in OMADA can participate in more than one agreement group and furthermore assume multiple roles. This approach allows OMADA to tailor the responsibilities of each replica to the individual performance capabilities of its server. While a replica on a powerful server, for example, may be part of several agreement groups and also act as executor, a replica on a slow server might only contribute to request ordering in a single agreement group.

Despite relying on multiple agreement groups that operate independently of each other, OMADA is nevertheless able to establish a total order on all client requests. To achieve this, OMADA splits the sequence-number space into partitions of equal size and statically maps one partition to each agreement group. In particular an agreement group g is responsible for assigning the sequence numbers $S_g = \{k \cdot |\mathcal{G}| + g | k \in \mathbf{N}\}$ by running separate instances of the PBFT [1] protocol; $\mathcal{G}$ denotes the set of all agreement groups. This approach of parallelizing agreement into multiple groups has the key advantage that the messages required for ordering a request only need to be exchanged between replicas of the respective agreement group, not between all agreement replicas in the entire system.

Knowledge about the number, composition, and sequence-number partitions of agreement groups, as well as the information which replicas act as executors, is static and available throughout the system. This, for example, allows a client to randomly select a group at start up which from then on will be responsible for handling all of the client's requests.

B. Scalable Ordering Based on Multiple Agreement Groups

In the following, we first present the basic OMADA protocol and then discuss specifics of OMADA including the coordination of agreement groups, executor checkpointing, fault handling, as well as optimizations. We use $\langle m \rangle \alpha_{i,\mathcal{R}}$ to denote a message m that has multiple recipients and is therefore authenticated with a MAC vector containing MACs between the sender i and each recipient j in the set $\mathcal{R}$. Besides, $\langle m \rangle \mu_{i,j}$ represents a message that is exchanged between sender i and a single recipient j and authenticated with a single MAC.

Basic Protocol. To use the replicated application, a client c sends a $\langle \text{REQUEST}, c, o, t \rangle \alpha_{c,\mathcal{A}}$ message to the leader of its agreement group. As the request will only be verified by members of this group, the authenticator of this message is limited to MACs for the group's agreement replicas $\mathcal{A}$. Apart from the command to execute o, the request also contains a client-local timestamp t that is incremented by the client on each operation. As agreement replicas store the timestamp t_c of the latest committed request of each client, the timestamp t allows them to detect and consequently ignore old requests.

Having received the request, the leader verifies that the message is authentic and then starts a new PBFT protocol instance to assign a unique sequence number s to the request; s is chosen as the lowest of the agreement group's unused sequence numbers (see Section III-A). Once the request is committed (i.e., at least $2f + 1$ agreement replicas have both verified the authenticity of the client request and confirmed the assignment of the sequence number), each agreement replica a sends an $\langle \text{EXECUTE}, v, s, q, a \rangle \alpha_{a,\mathcal{E}}$ message to all executors $\mathcal{E}$. In addition to the client request q, this message also comprises the information in which view v the request has committed on the local replica. This knowledge is later used to determine the replica the client should contact for a subsequent request.

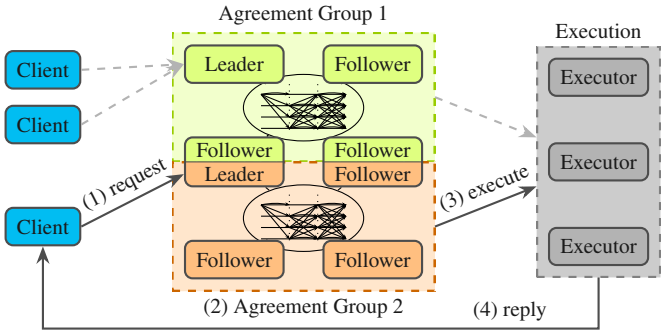


Figure 2. Overview of the OMADA architecture relying on multiple, possibly overlapping, agreement groups. To invoke an operation at the application, a client (1) sends a request to one of the agreement groups, which then (2) orders the request using PBFT and (3) forwards it to a set of executors. Having processed the request, (4) the executors return their results to the client.

An executor only accepts an EXECUTE if the message is authentic and its sender i is indeed an agreement replica of the group responsible for assigning sequence number s. Before executing the corresponding request, an executor first waits until having obtained $f+1$ matching EXECUTEs from different agreement replicas, as this proves that at least one correct replica has committed the request. As the same request may be committed in different views on different replicas, an executor ignores the view information when comparing EXECUTEs.

Although EXECUTEs potentially arrive in a nondeterministic pattern, executors process requests in the order of their sequence numbers, leaving no gaps between sequence numbers. Similar to agreement replicas, executors manage a timestamp t_c for each client, which is the timestamp of the latest request of a client c the executor has processed. To prevent multiple invocations of the same request, the execution of a request from a client c with a timestamp $t \leq t_c$ consists of a no-op.

Having processed a request, an executor e sends the result r in a $\langle \text{REPLY}, c, t, e, r, v \rangle \mu_{e,c}$ message to the client c; v represents the current view of the group that ordered the request and is used to update the client's leader information. To suppress the effects of faulty replicas, an executor selects v to be the $f+1$ highest view it has learned from replicas of the group. A client accepts a result after having received $f+1$ REPLYs with matching r from different executors as this guarantees that at least one of the messages originates from a correct executor.

Coordination of Agreement Groups. Agreement groups in OMADA operate independently of each other and therefore possibly advance at different speeds. As a result, one group may for example already have committed a request for sequence number s while another group has not yet reached sequence number $s - 1$. To ensure liveness in such cases, OMADA allows groups to detect that they have fallen behind by receiving notifications from executors when requests with higher sequence numbers become ready for processing.

Having detected a gap in the sequence of executable requests, an executor e broadcasts a $\langle \text{FLUSH}, s, s_{exec}, e \rangle \alpha_{e,\mathcal{Z}}$ message to all agreement replicas $\mathcal{Z}$ in the system notifying them about s, the highest sequence number for which the executor has collected $f+1$ matching EXECUTEs, and s_{exec}, the sequence number of the last request it has executed. The broadcast is repeated periodically until the gap is closed. By combining the information contained in FLUSHes from different executors, agreement replicas are able to reliably determine the overall system progress. For this purpose, each replica calculates $s_{progress}$ to be the $f+1$ highest sequence number s the replica has learned from different executors.

Based on a comparison of $s_{progress}$ with the latest sequence number s_g for which the agreement process has been started, replicas of an agreement group g can determine whether their group has fallen behind in relation to other groups. If this is the case, the leader of a group starts new PBFT instances for all of the group's sequence numbers between s_g and $s_{progress}$, proposing either a client request (if available) or a no-op. Consequently, the sequence-number gap that temporarily prevents executors from processing further requests will eventually be closed, enabling the overall system to make progress again.

To ensure liveness in the presence of a faulty leader, the followers of a group monitor the leader's behavior. If the leader fails to start the required instances within a certain period of time, the followers initiate a view change to replace the leader. Apart from that, to tolerate message losses agreement replicas retransmit EXECUTEs for sequence numbers higher than s_{exec} when receiving repeated FLUSH messages from an executor.

Executor Checkpoints. With EXECUTEs not necessarily arriving in the order of their sequence numbers, executors may need to buffer them. To guarantee a bounded buffer size, an executor uses a sliding window of size $W = 2 * cp_{interval}$ [16] and only stores EXECUTEs with numbers between s_{low} and $s_{low} + W$. To advance the window, in intervals of $cp_{interval}$ each executor e creates and stores a checkpoint cp of the application state, the latest client timestamps, and the latest reply it has sent to each client. Furthermore, the executor broadcasts a $\langle \text{CHECKPOINT}, s, D(cp), e \rangle \alpha_{e,\mathcal{Z} \cup \mathcal{E}}$ message to all agreement replicas and executors; s is the sequence number of the latest request the executor has processed prior to the snapshot and $D(cp)$ denotes a hash of the checkpoint.

When an executor receives $f+1$ matching CHECKPOINTs from different executors for a sequence number $s > s_{low}$, the checkpoint becomes stable. At this point, the executor sets the start of its local window to sequence number s and discards all EXECUTEs and checkpoints before $s - cp_{interval}$. If an executor has fallen behind, advancing the window can result in client requests being skipped. To ensure a consistent application state in such scenarios, an executor first obtains a full checkpoint with matching sequence number and hash from another executor before continuing to process further requests.

Besides guaranteeing execution-stage progress, CHECKPOINTs also enable agreement groups to perform garbage collection of internal messages. For this purpose, an agreement replica notifies its local PBFT node about stable checkpoints.

Fault Handling. OMADA is able to tolerate up to f faulty replicas per agreement group and a maximum of f faulty executors. In heterogeneous settings where some replicas assume multiple roles, the failure of a replica can affect more than one component. Relying on PBFT for request ordering has the key advantage that for many fault scenarios, OMADA does not need to provide additional mechanisms, as they are already handled by the agreement protocol. In particular, this applies to situations in which the leader of an agreement group fails to live up to its responsibilities and is consequently replaced as the result of a view change requested by its followers.

If a client issues a request but does not get a result within a predefined period of time, the client sends the request to both all replicas of its agreement group as well as all executors. This way, agreement replicas learn about the problem and if necessary can initiate a view change or retransmit the EXECUTE for a committed request to handle cases in which previous messages to executors have been lost due to network problems. On the other hand, executors resend the corresponding reply (if available) when receiving a request directly from a client.

Optimizations. OMADA supports common BFT-system optimizations [1] such as payload hashes and batching (i.e., ordering multiple requests in the same PBFT instance); if batching is applied, clients use the individual batch sizes of agreement groups as relative weights when randomly selecting a group. In addition, there is room for OMADA-specific optimizations if a replica assumes more than one role: First, messages to multiple receivers (e.g., FLUSHes) need to be sent only once to each server. Second, if the same replica acts both as an agreement replica as well as an executor, a request becomes ready for processing as soon as it has been committed locally; the executor does not have to wait for an external proof in the form of $f+1$ matching EXECUTEs. As a result, it is sufficient for agreement replicas to only send EXECUTEs to those executors whose replicas are not part of the same group.

C. Supporting Heterogeneous Servers

To effectively exploit the resources available in heterogeneous settings, OMADA statically tailors the responsibilities of each replica to the individual performance capabilities of its server before startup. In the following, we describe the systematic approach to determine the assignment of roles to replicas we use for this purpose: First, we assess the performance capabilities of each server. Next, we estimate how many resources to reserve for the agreement stage compared with the execution stage. Then, we rely on an integer linear program to determine the number of agreement groups as well as the mapping of roles to replicas. Finally, in the last step we define an individual batch size for each agreement group.

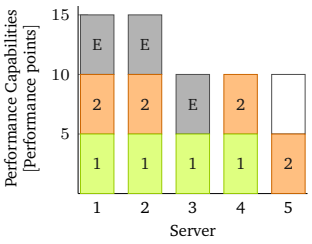
Assessing the Performance Capabilities of Servers. Prior to being able to assign replica roles, we first need to identify the differences in performance between the servers involved. To achieve this, on each server, we execute a small benchmark that measures the number of MACs the server can calculate per second. This empirical approach has two key advantages: First, it assesses the individual performance capability of a server

based on the operation that is the dominant factor with regard to OMADA's overall computing-resource usage. Second, the approach also provides reliable results in situations in which the actual performance of a server differs from the targeted performance, as it is often the case in cloud environments [9].

As our assignment algorithm operates with relative performance values, we translate the measured performance numbers into *performance points* reflecting the differences between servers, beginning with 10 points for the slowest server. To illustrate this step, Figure 3 shows an example for a heterogeneous setting with five servers in which the two fast servers are able to perform 50 % more MAC calculations per second than the three slow servers. Consequently, we assign 15 and 10 points to the fast and slow servers, respectively.

Relative Costs for Agreement and Execution. To estimate the relative amount of resources OMADA needs to reserve for agreement and execution, we compare the number of MACs each stage computes per client request during normal-case operation. Ordering requests in batches of size b, an agreement replica must perform $1 + \frac{12f+1}{b}$ MAC calculations per request: 1 for verifying the authenticity of the request, $\frac{10f}{b}$ for ordering it with PBFT, and $\frac{2f+1}{b}$ for sending EXECUTEs to the executors. In contrast, an executor only calculates $\frac{f+1}{b} + 1$ MACs per request: $\frac{f+1}{b}$ for verifying the EXECUTEs and 1 for authenticating the reply to the client. For $f = 1$ and a typical batching factor of $b = 10$, this means that participating in all agreement groups requires about twice as many computing resources as assuming the role of an executor (e.g., 10 versus 5 performance points for Server 1 in the example in Figure 3). A similar ratio applies in the optimized case where an agreement group does not need to send EXECUTE messages due to one of its members being collocated with an executor.

Assignment of Roles to Replicas. Having determined the individual capabilities of servers as well as the relative costs for agreement and execution, we can derive the mapping of roles to replicas. As common methods such as greedy or knapsack algorithms are either unable to always find optimal solutions or to limit the number of groups, we formulate the problem of mapping roles to replicas as an integer linear program [17], as shown in Figure 3. In a nutshell, this approach allows us to automatically examine all possible distributions of agreement groups with $3f + 1$ replicas across the servers available in order to find a configuration that maximizes performance. By specifying a number of constraints, we ensure that the selected configuration provides certain properties: First, the configuration allocates an identical amount of resources to all members of the same group to ensure that performance remains stable across view changes (see Figure 3, the sum in Line 7–8). Second, it respects the individual performance limits of each server (Line 7–8). Third, it places executors on the $2f + 1$ most powerful servers, thereby increasing the number of agreement groups that are able to benefit from collocation with an executor (Line 8–9). Fourth, it does not make use of more than a predefined number of agreement groups to keep the coordination overhead low (Lines 10–11).



Var.	Description
$\mathcal{R}$	Set of all servers
$\mathcal{E}$	Set of executors (i.e., the 2f+1 servers with the most perf. points)
l_r	Perf. points of server r
G_{max}	Maximum number of groups
$\mathcal{P}$	All possible group permutations
x_g	Perf. points for group $g \in \mathcal{P}$
o_g	Number of executors without a collocated agreement replica of g

Group Assignment

1 **Variables**
2 $w = \max_{r \in \mathcal{R}}(l_r)$
3 $cf_g = \frac{1}{1+o_g * 0.15}$
4 $\mathcal{P} = \{p | p \subset \mathcal{R} \wedge |p| = 3f + 1\}$
5 $|\mathcal{P}| = \binom{|\mathcal{R}|}{3f+1}$

6 **Constraints**
7 $\forall r \in \mathcal{R}, r \notin \mathcal{E}: \sum_{g \in \mathcal{P}, r \in g} x_g \leq l_r$
8 $\forall r \in \mathcal{E}: x_e + \sum_{g \in \mathcal{P}, r \in g} x_g \leq l_r$
9 $\sum_{g \in \mathcal{P}} (0.5 x_g) - 0.5 \leq x_e$

10 $\forall g \in \mathcal{P}: x_g + w * x_{gp} \leq w$
11 $\sum_{g \in \mathcal{P}} x_{gp} \geq |\mathcal{P}| - G_{max}$

12 **Optimization target**
13 $\max(\sum_{g \in \mathcal{P}} cf_g * x_g)$

Figure 3. Integer linear program for assigning roles to replicas. To account for efficiency gains achieved by collocating an agreement replica with an executor, the program weights agreement groups using a cost factor cf_g that reflects an empirically determined cost increase of 15 % per separate executor.

Obeying these constraints, the integer linear program assigns individual performance points to each group in the set of possible group-to-server mappings. Based on this result, we can compile the final OMADA configuration by including all groups that received at least one performance point. Furthermore, with each group id representing a particular group-to-server mapping, the result also directly contains the placement of groups. In case the integer linear program produces multiple solutions, we select the solution with the lowest number of groups and the smallest relative performance-point differences between groups to minimize the coordination overhead.

Selection of Batch Sizes. To implement performance differences between agreement groups, we define the batch size for each group individually. For a group g, we calculate the batch size by multiplying its performance points with a cost factor cf_g (see Figure 3) and normalize the result such that the weakest group uses a batch size of 10. For the configuration in Figure 3, this for example leads to normalized and rounded batch sizes of $5 * 1 \rightarrow 12$ and $5 * \frac{1}{1+1*0.15} \rightarrow 10$ for Group 1 and 2, respectively, which reflects the fact that the executor on Server 3 does not have a collocated replica of Group 2. Less powerful groups process fewer requests to be able to keep up with the more powerful groups. This is necessary as all groups have to handle the same amount of sequence numbers.

IV. EVALUATION

In this section, we evaluate OMADA based on a coordination service that relies on our protocol for fault tolerance and comprises a similar interface as ZooKeeper [15]. Coordination services are key building blocks of today's data-center infrastructures as they allow processes of distributed applications to cooperate, for example, by reliably exchanging small chunks of data. As a consequence of being essential for the well-functioning of other applications, it is crucial for coordination services to provide resilience against a wide spectrum of faults.

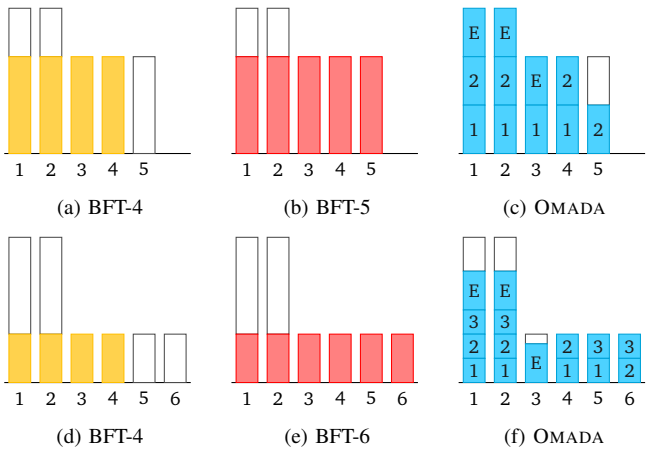
A. Environment

To evaluate OMADA in comparison to existing approaches, we enable our current prototype implementation to also apply other protocols for replication. Using this prototype, we repeat all our experiments with the following settings, all of which are configured to be able to tolerate one Byzantine fault:

- *BFT-4* makes use of the minimum number of replicas required for Byzantine fault tolerance (i.e., four replicas) and thereby represents the typical setting found in most state-of-the-art BFT systems. To order client requests, the BFT-4 implementation executes the PBFT protocol [1].
- *BFT-5* and *BFT-6* are variants of BFT-4 with five and six replicas, respectively, each running on a dedicated server. These settings allow us to study the effects of introducing additional resources into a traditional BFT system.
- OMADA relies on our novel BFT protocol presented in Section III and is distributed across up to six servers.

In order to be able to investigate the influence of heterogeneity, we conduct our experiments using two different settings of servers with non-uniform performance capabilities, as illustrated in Figure 4. In this context, we distinguish between two categories of machines: *fast* servers and *slow* servers; the difference in performance achieved varies between experiments and will therefore be explained in later sections. All servers use Ubuntu 14.04 LTS as operating system along with OpenJDK 7u121 and are connected via switched Gigabit Ethernet.

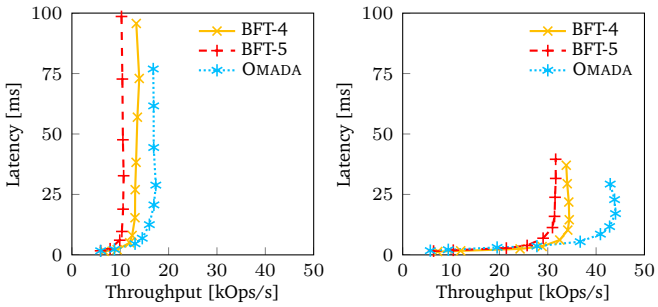
For each experiment, we vary the number of clients writing data to the coordination service in chunks of typical sizes of 128 bytes. Besides, we have also conducted experiments evaluating read operations, but we omit these results due to limited space and because they offer similar insights as the write results. To generate the workloads, we execute the clients on a separate server and distribute them across the available agreement groups in a way so that more powerful groups handle more clients. During an experiment, each client runs in a closed loop, that is, it only sends a new request after having obtained a stable reply for its previous one. Each data point presented in the following represents the average over 5 runs.



(a) BFT-4 (b) BFT-5 (c) OMADA

(d) BFT-4 (e) BFT-6 (f) OMADA

Figure 4. System configurations for the two heterogeneous settings used in the evaluation comprising 5 servers (top) and 6 servers (bottom), respectively.



(a) Batching disabled (b) Batching enabled

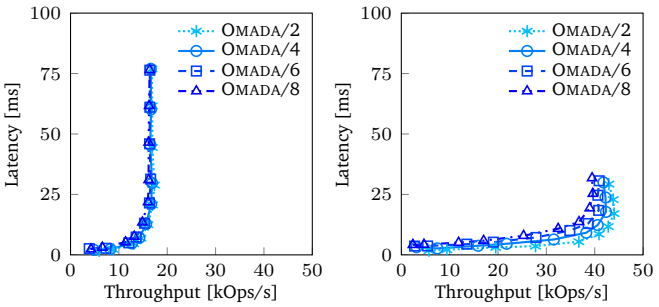
Figure 5. Results for the heterogeneous setting with 2 fast and 3 slow servers.

B. Exploiting Additional Computing Resources

Our first heterogeneous setting comprises two fast and three slow servers (see Figures 4a–4c). The fast servers are equipped with Intel Xeon E5645 CPUs (2.4 GHz) and 32 GB RAM, whereas the slow servers have Intel Xeon E5520 CPUs (1.6 GHz) and 8 GB RAM. Based on the rate of MAC calculations per second, a slow server in this setting achieves about two thirds of the performance of a fast server. For this environment, OMADA's group assignment procedure creates the configuration we already used as example to explain the procedure in Section III-C. It comprises two agreement groups with batch sizes of 12 and 10, respectively. For a fair comparison, we configure BFT-4 and BFT-5 to use a batch size of 11, which is the average batch size of the two OMADA groups. To examine the effects of the optimization, we conduct all experiments with both batching disabled and enabled.

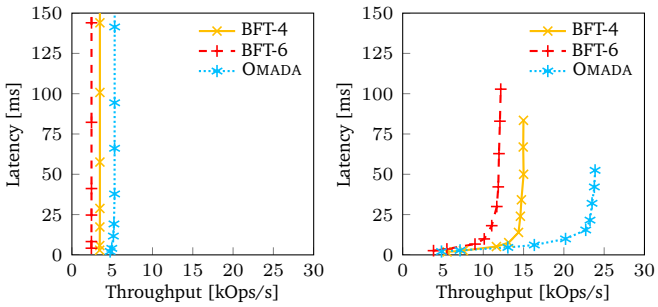
Figure 5 presents the measured latency and throughput for this experiment. All three systems achieve low latency until reaching saturation, at which point latency rises quickly. In general, the latency of BFT-5 is higher than that of BFT-4. This is caused by the larger quorum sizes and thus the additional messages that are necessary to include the fifth server, as well as the larger MAC authenticators which grow in size and require an additional MAC calculation. Compared with BFT-4, the latency of OMADA is slightly higher which can be attributed to the increased coordination overhead necessary to manage the two agreement groups. However, unlike BFT-4, OMADA is able to sustain low latency at higher throughputs.

With regard to the maximum throughput achievable, our results confirm the key advantage of the batching optimization, which enables all three systems to provide a significantly higher performance. Nevertheless, due to not being able to benefit from the additional server, the maximum throughput of BFT-5 is still about 8% lower than the maximum throughput of BFT-4; without batching the difference is about 23%. In contrast, OMADA exploits the additional computing resources offered by the fifth server and therefore compared with BFT-4 achieves an increase in maximum throughput of 25% when batching is disabled. With batching enabled, the improvement over BFT-4 is almost 28%. This is mainly a result of the reduced group-coordination overhead, which in this case only has to be paid once per batch and no longer once per request.



(a) Batching disabled (b) Batching enabled

Figure 6. Results for OMADA with different group counts (5-server setting).



(a) Batching disabled (b) Batching enabled

Figure 7. Results for the heterogeneous setting with 2 fast and 4 slow servers.

C. Assessing the Costs of Groups

In order to evaluate the group-coordination overhead in OMADA in more detail, we use the same setup as in the previous experiment but vary the number of agreement groups by splitting each of the two existing groups shown in Figure 4c into up to four smaller ones. This yields four configurations comprising between two and eight agreement groups which in the following are referred to as OMADA/2 to OMADA/8.

The results in Figure 6 show that the overhead for operating additional agreement groups is small but measurable. With agreement groups in OMADA being largely independent of each other, having fewer groups has the advantage that it becomes less likely that requests of one group need to wait at an executor until requests with lower sequence numbers of another group become ready for execution. As a consequence, OMADA/2 with its two agreement groups provides lower latency than OMADA/8 with eight agreement groups. A similar effect as for latency can also be observed with regard to throughput: The maximum throughput of OMADA/2 without batching for example is about 6% higher than the maximum throughput of OMADA/8; when batching is enabled, the difference between these two configurations is 11%. In summary, the results of this experiment confirm our strategy of selecting the configuration with the lowest number of agreement groups in cases in which OMADA's group assignment procedure proposes multiple possible solutions (see Section III-C). Furthermore, our observations are also consistent with the intuition that the price of an increased group-coordination overhead should only be paid if a configuration with a higher number of agreement groups is actually able to exploit further computing resources, for example, in the form of additional servers.

D. Analyzing the Effects of Heterogeneity

For our third experiment, we use a heterogeneous environment that differs from the setting of the previous experiments, which enables us to study the adaptability of OMADA. As shown in Figures 4d–4f, the systems now have an additional server at their disposal. Furthermore, by limiting the number of MAC calculations a slow server is allowed to perform per second, we create a scenario in which a slow server only achieves a third of the performance of a fast server. In practice, such performance differences between replicas can be the result of not only incorporating servers with different capabilities but also relying on heterogeneous replica implementations to minimize the probability of common mode failures. For this purpose, the replicas of a system for example may make use of different programming languages or operating systems in order to reduce the fault dependency between them [18]. In the heterogeneous environment of this experiment, OMADA has three agreement groups that are distributed across servers as shown in Figure 4f. As all groups use a batch size of 10 we select the same batch size for BFT-4 and BFT-6.

The results in Figure 7 show that as an effect of the reduced amount of computing resources, the maximum throughputs achieved in this experiment are lower than the maximum throughputs in previous experiments. In particular, both BFT-4 and BFT-6 are unable to utilize most of the resources available on the fast servers due to being limited by the slow servers. With a decreased maximum throughput of 30% (without batching) and 19% (with batching) compared with BFT-4, BFT-6 performs significantly worse than its counterpart BFT-5 in Section IV-B, which confirms the system's lack of scalability. OMADA, on the other hand, not only benefits from the additional servers but also utilizes a large part of the computing resources on the fast servers by enabling their replicas to act as executors and to furthermore participate in all three agreement groups. This way, when batching is disabled OMADA achieves a maximum throughput that is 53% higher than the maximum throughput of BFT-4 for this experiment. As in previous experiments, enabling the batching optimization has a positive impact on both the overall maximum throughput and the improvement over BFT-4, which in this experiment is 59%.

V. Related Work

Yin et al. [12] proposed a BFT system architecture (in the following referred to as SAfE) that separates agreement from execution and comprises a dedicated cluster of replicas for each of the two stages. OMADA builds on this idea by splitting the responsibilities for ordering and executing requests into different roles and allowing each replica to assume one or more of these roles depending on its performance capabilities. As a consequence, both SAfE and OMADA need to provide the agreement stage with means to prove to the execution stage that a request has been committed. In SAfE, the agreement

cluster for this purpose sends PBFT-internal messages to the execution cluster, thereby (1) creating a close coupling between both stages and (2) requiring agreement-protocol messages to be authenticated with additional MACs in order to be verifiable by the execution cluster. In contrast, OMADA cleanly decouples the agreement protocol from the execution protocol (i.e., the transmission of EXECUTEs) to keep the authentication cost for agreement messages low. Furthermore, OMADA saves further network and computing resources by suppressing an agreement group's EXECUTEs if an executor is collocated with an agreement replica of the group.

UpRight [16] goes one step further than SAfE and, besides agreement and execution, relies on a third stage that is responsible for receiving and buffering client requests. Using this stage, the system can forward large requests directly to the execution while performing the agreement on their hashes, thereby reducing the load on the agreement cluster at the cost of increased latency. Similar to SAfE and PBFT, the agreement stage of UpRight only comprises $3f + 1$ replicas and consequently is not able to benefit from additional servers.

Kapritsos and Junqueira [19] outlined a general approach to partition the agreement workload in order to improve the scalability of replicated systems. In particular, they presented a crash-tolerant protocol that, similar to OMADA, is able to assign different parts of the sequence-number space to different replica groups. Having been designed to provide resilience against crashes, unlike OMADA, the protocol however cannot ensure liveness in environments in which replicas may fail in arbitrary ways. Furthermore, their approach does not address replicated systems that consist of heterogeneous servers.

The idea of using multiple replicas to independently order requests that are then merged into a single ordered request stream has been explored in various ways in the context of crash fault tolerance. In the accelerated ring protocol developed by Babay and Amir [20] the replicas pass on a single token after proposing several requests while compensating for network latency. Aguilera and Strom [21] propose an algorithm to deterministically merge multiple message streams primarily based on the timestamps of individual messages. Mencius [22], a crash-tolerant protocol for wide-area networks, evenly partitions the sequence numbers across all replicas.

COP [7] and Sarek [8] parallelize the handling of agreement-protocol instances within each replica of a BFT system in order to effectively utilize multi-core servers. Focusing on the internal structure of a replica, these approaches are orthogonal to the replication scheme presented in this paper and could therefore also be applied to increase performance in OMADA.

The few works that have investigated heterogeneity in the context of BFT systems [3], [10] use heterogeneous execution-stage implementations to reduce the probability that a single fault causes multiple replica failures. OMADA, on the other hand, deals with the consequences of heterogeneity at the level of the entire system. This approach enables OMADA to exploit performance capabilities that have remained unused so far, following the principle that it is better to harness the differences between replicas than to try to compensate them.

VI. CONCLUSION

OMADA is a BFT protocol that is able to use additional servers by partitioning the agreement stage into multiple largely independent groups. In environments comprising servers with heterogeneous performance capabilities, OMADA tailors the distribution of the agreement groups to the set of servers available in order to exploit the individual performance capabilities of each server. Our evaluation has shown that in contrast to existing systems OMADA is able to benefit from additional computing resources, and that our approach is particularly effective in heterogeneous settings with a significant performance difference between fast and slow servers.

Acknowledgments: This work was partially supported by the German Research Council (DFG) under grant no. DI 2097/1-2 ("REFIT").

REFERENCES

[1] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. of OSDI '99*, pp. 173–186.
[2] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, 1980.
[3] M. Castro, R. Rodrigues, and B. Liskov, "BASE: Using abstraction to improve fault tolerance," *ACM Trans. on Computer Systems*, vol. 21, no. 3, pp. 236–269, 2003.
[4] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung, "Spin one's wheels? Byzantine fault tolerance with a spinning primary," in *Proc. of SRDS '09*, pp. 135–144.
[5] P.-L. Aublin, S. B. Mokhtar, and V. Quéma, "RBFT: Redundant Byzantine fault tolerance," in *Proc. of ICDCS '13*, pp. 297–306.
[6] A. Bessani, J. Sousa, and E. E. P. Alchieri, "State machine replication for the masses with BFT-SMaRt," in *Proc. of DSN '14*, pp. 355–362.
[7] J. Behl, T. Distler, and R. Kapitza, "Consensus-oriented parallelization: How to earn your first million," in *Proc. of Middleware '15*, pp. 173–184.
[8] B. Li, W. Xu, M. Z. Abid, T. Distler, and R. Kapitza, "SAREK: Optimistic parallel ordering in Byzantine fault tolerance," in *Proc. of EDCC '16*, pp. 77–88.
[9] Z. Ou, H. Zhuang, A. Lukyanenko, J. K. Nurminen, P. Hui, V. Mazalov, and A. Ylä-Jääski, "Is the same instance type created equal? Exploiting heterogeneity of public clouds," *IEEE Trans. on Cloud Computing*, vol. 1, no. 2, pp. 201–214, 2013.
[10] T. Distler, R. Kapitza, and H. P. Reiser, "State transfer for hypervisor-based proactive recovery of heterogeneous replicated services," in *Proc. of SICHERHEIT '10*, pp. 61–72.
[11] F. Junqueira, R. Bhagwan, A. Hevia, K. Marzullo, and G. M. Voelker, "Surviving Internet catastrophes," in *Proc. of ATC '05*, pp. 45–60.
[12] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin, "Separating agreement from execution for Byzantine fault tolerant services," in *Proc. of SOSP '03*, pp. 253–267.
[13] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel, "CheapBFT: Resource-efficient Byzantine fault tolerance," in *Proc. of EuroSys '12*, pp. 295–308.
[14] T. Distler, C. Cachin, and R. Kapitza, "Resource-efficient Byzantine fault tolerance," *IEEE Trans. on Comput.*, vol. 65, no. 9, pp. 2807–2819, 2016.
[15] P. Hunt, M. Konar, F. Junqueira, and B. Reed, "ZooKeeper: Wait-free coordination for Internet-scale systems," in *Proc. of ATC '10*, pp. 145–158.
[16] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche, "UpRight cluster services," in *Proc. of SOSP '09*, pp. 277–290.
[17] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.
[18] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "Analysis of operating system diversity for intrusion tolerance," *Software—Practice & Experience*, vol. 44, no. 6, pp. 735–770, 2014.
[19] M. Kapritsos and F. P. Junqueira, "Scalable agreement: Toward ordering as a service," in *Proc. of HotDep '10*, pp. 7–12.
[20] A. Babay and Y. Amir, "Fast total ordering for modern data centers," in *Proc. of ICDCS '16*, pp. 669–679.
[21] M. K. Aguilera and R. E. Strom, "Efficient atomic broadcast using deterministic merge," in *Proc. of PODC '00*, pp. 209–218.
[22] Y. Mao, F. P. Junqueira, and K. Marzullo, "Mencius: Building efficient replicated state machines for WANs," in *Proc. of OSDI '08*, pp. 369–384.