GAIA: Strengthening the Reliability of Datacenter Computing with Fast and Scalable Distributed Consensus

Abstract:

To deal with the rapidly increasing volume of data, more and more software applications run within a datacenter containing numerous computers. To harness the massive computing resources, applications are deployed with two major types of infrastructures: schedulers and virtual machines. These infrastructures have brought many benefits, including improving resource utilization, balancing loads, and saving energy. Unfortunately, as an application runs on more computers, minor computer failures will occur more likely and can turn down the entire application, causing severe disasters such as the 2015 NYSE trading halts and recent Facebook outages. Existing infrastructures lack support to ensure high-availability for applications.

This GAIA project takes a holistic methodology to greatly improve application availability with three objectives. First, we will create a fast, scalable distributed consensus protocol for general applications. Distributed consensus provides strong fault-tolerance: it replicates the same application on different computers and always enforces same inputs on these computers, as long as a majority of computers are alive. An open challenge is that traditional consensus protocols are too slow mainly because their messages go through OS kernels and software TCP/IP layers. We tackle this challenge by creating a fast consensus protocol with an ultra-fast kernel bypassing technique called Remote Direct Memory Access (RDMA). Our preliminary results published in [SOSP '15] show that our protocol can support diverse, unmodified applications, and our latest protocol was 20X~31X faster than traditional protocols even when running on 35X more computers.

Second, we will construct the first fault-tolerant scheduler by integrating our protocol with popular datacenter schedulers. To seamlessly achieve resource allocation and application replication, we propose a new replication-aware resource allocation workflow for our scheduler. Preliminary results published in [APSys ’16] show that this scheduler can efficiently support Memcached, a widely used key-value store.

Third, we will make our protocol and virtual machines (VM) form a mutual-beneficial eco-system. This eco-system not only leverages the VM hypervisor layer to automatically replicate applications, but it also introduces a new VM live migration approach for computer load balance. To migrate application execution states to remote computers, prior live migration approaches incur substantial application down time and resource consumption on local computer. Our new approach need only migrate a consensus leadership, which consumes almost zero time and resource.

We believe that our expertise on building reliable distributed systems and our preliminary results will help us achieve all the three objectives. By greatly strengthening the availability of

datacenter applications, this project will benefit almost all computer users and software vendors, including many financial platforms in HK. This project will also advance various datacenter techniques (e.g., migration) and attract researchers to build more reliable infrastructures.

Long term impact:

To deal with the rapidly increasing volume of data, more and more software applications run within a datacenter containing numerous computers. Many applications are mission-critical and naturally demand high reliability and performance, including financial platforms, social network platforms, military services, and medical services. To harness the massive computing resources, applications are deployed with two major types of infrastructures: schedulers and virtual machines. These infrastructures have brought many benefits, including improving resource utilization, balancing loads, and saving energy.

Unfortunately, as an application runs on more computers, minor computer failures will occur more likely and can turn down the entire application, if the failure computer runs a critical computation. For instance, recently, both New York Stock Exchange (NYSE) and Nasdaq have experienced outages of their whole site or delays of IPO events due to minor machine errors. In addition, social network applications such as Facebook tend to be online 24-7, but minor computer failures have turned down the whole Facebook site for several times in the last few years. All these outage events have led to huge money lost.

A key problem of these disasters is that datacenter infrastructures lack high availability support for general applications. As a result, although some applications have built ad-hoc replication approaches to improve their own availability, most other applications still suffer.

This proposed GAIA project takes a holistic methodology to tackle this problem. It first plans to build a fast, scalable replication protocol, it then integrates this protocol into two major infrastructures, potentially benefiting all applications. We envision significant impacts for this project in different terms.

In the near term, Falcon (Objective 1) can largely improve both the scale and performance of many replication systems. For instance, a notable key-value store system called Scatter deploys 8~12 replicas in each Paxos group, and now it can deploy hundreds of replicas in each group and achieves much better performance. Overall, a fast, scalable, and general service, Falcon may significantly promote the deployments of Paxos and improve both the performance and fault-tolerance of various systems in datacenters.

In the intermediate term, by realizing Objective 2 and 3, we will greatly strengthen the availability of datacenter applications and benefit almost all computer users and software vendors, including many financial platforms in HK. Because HK has lots of financial platforms which naturally desire high availability in their operational hours, this GAIA project can bring

practical benefits to these platforms and avoid horrible outages such as the 2015 NYSE trading halts.

In the long term, we anticipate that this project will advance various datacenter techniques (e.g., live migration) and attract researchers to build more reliable infrastructures. As datacenter emerges to be a "giant computer", a future datacenter OS for such a computer will gradually come up. Therefore, consensus protocols, schedulers, and virtual machines will become essential components for this OS, and the outcomes of this GAIA project will eventually be adopted in a future datacenter OS.

1 Research Background

This section presents the background of consensus (§1.1) and datacenter computing infrastrutures (§1.2), motivation of objectives (§1.3), others' related work (§1.4), and PI's related work (§1.5).

1.1 Paxos Consensus

Consensus protocols (typically, PAXOS [47, 48, 50, 62]) play a key role in datacenters, including ordering services [33, 44, 52], leader election [11, 20], and fault-tolerance [27, 35, 45]. A PAXOS protocol replicates the same application on a group of computers (or *replicas*) and enforces the same order of inputs for this application, as long as a majority of replicas are still alive. Therefore, PAXOS tolerates various faults, including minor replica failures and packet losses.

PAXOS is widely served in many systems. For instance, Scatter [33] runs 8$\sim$12 replicas in a PAXOS group to order client requests, and it lets replicas reply requests in parallel. A bigger group size will improve Scatter throughput. Moreover, state machine replication (SMR) systems [27, 35, 45] use PAXOS to improve the availability of server applications (e.g., MySQL [12]).

Unfortunately, the group size of existing PAXOS protocols can hardly go up to a dozen because their consensus messages go through OS kernels and software TCP/IP layers (a round-trip takes about 200 μs), causing the consensus latency to increase almost linearly to group size [11, 27, 33].

To address this PAXOS performance problem, Remote Direct Memory Access (RDMA) (e.g., Infiniband [6]) is a promising solution due to its commonplace in datacenters and and its decreasing prices. An RDMA round-trip takes only about 3 μs [53]. This ultra low latency not only comes from its kernel bypassing feature, but also its dedicated network stack implemented in hardware. Therefore, RDMA is considered the fastest kernel bypassing technique [43, 53, 59]; it is several times faster than software-only kernel bypassing techniques (e.g., DPDK [3] and Arrakis [58]).

1.2 Datacenter Computing Infrastrutures

To deal with the rapidly increasing volume of data, more and more applications are deployed in a datacenter with two complementary types of infrastrutures. The first type is schedulers [13, 19, 38, 39, 41, 63, 64, 72]. Many applications are mission critical and demand both high performance and availability, including financial trading, health care, social networks, and military applications. For instance, a high-frequency trading platform must be highly-available during operation hours, and adding merely hundreds of μs to its latency means big money lost [37]. Another instance is social networks: minor computer errors in Facebook cluster led to several outages in recent years [5].

Although existing schedulers themselves have are made available with PAXOS (e.g., [38]), their applications are not. Therefore, if an application crashes or hardware errors occur, these schedulers have to reschedule applications, leading to substantial application unavailability.

The second infrastruture type is VM [14, 16, 46, 57, 65]. VM abstracts away the heterougous physical computing resources, making computing resources easy to isolate, utilize, and balance loads (e.g., via live migration [22, 55]). Although some application fault-tolerance approaches such as primary-backup [28], these approaches still often consume prohibitive resources (e.g., CPU and network bandwith) and have weaker availability guarantee than PAXOS.

Schedulers and VMs are largely complementary infrastrutures depending on application requirements such as performance and security. Many schedulers use lightweight containers [2, 8] for isolation, while many other cloud deployments (e.g., Amazon EC2) simply use VMs without schedulers. Therefore, this proposal strenghtens the two infrastrutures respectively.

1.3 Motivation of Objectives

The proposed objectives stem from two research problems in datacenter computing. First, despite the core role and wide deployments of PAXOS, it suffers from high consensus latency and poor scalability. Therefore, **Objective 1** addresses this problem by leveraging RDMA to create a fast, scalable consensus protocol. Second, although many datacenter applications demand high availability, existing infrastrutures lack such support. To benefit general applications, **Object 2 and 3** take a holistic methodology to integrate our protocol with two major infrastrutures

1.4 Related Work by Others

Various Consensus Protocols. There are a rich set of PAXOS algorithms [47, 48, 50, 54, 62] and implementations [20, 21, 27, 50]. PAXOS is notoriously difficult to be fast and scalable [33, 44, 52]. Since consensus protocols play a core role in datacenters [1, 38, 71] and worldwide distributed systems [23, 49], various works have been conducted to improve specific aspects of consensus protocols, including commutativity [54], understandability [48, 56], and verifiable rules [34, 69].

To make PAXOS's throughput scalable (i.e., more replicas, higher throughput), various systems leverage PAXOS as a core building block to develop advanced replication approaches, including partitioning program states [17, 33], splitting consensus leadership [18, 49], and hierarchical replication [33, 44]. Theses approaches have shown to largely improve throughput. However, the core of these systems, PAXOS, still faces an unscalable consensus latency [33, 44, 52]. By using GAIA as a building block, these prior systems can scale even better.

Fault-tolerance in schedulers. Datacenter schedulers [13, 19, 38, 39, 41, 63, 64, 72] are widespread because they can support diverse applications (e.g., Hadoop [36], Dryad [40], and key-value stores [61]). These existing systems mainly focus on high availability for themselves by replicating their own important components, or focus on fault-recovery of applications [72]. To the best of our knowledge, no existing schedulers provide fast and general high-availability service to applications.

Fault-tolerance in VM. Two approaches, primary-backup and live migration, exist for improving application reliability in VM. Primary-backup uses the hypervisor layer to record application execution state changes in a primary VM and frequently propagate the changes to a backup VM on another computer. Live migration is invoked for both computer load balance and handling failures, and it uses a similar hypervisor technique as primary-backup. Despite much clever effort, both these two approaches consume substantial application down time (e.g., 8 seconds in vMotion [55]) and network bandwidth.

RDMA techniques. RDMA has been realized in various types of datacenter networks, including Infiniband [6], RoCE [10], and iWRAP [7]. RDMA has been leveraged in many software systems to improve different performance aspects, including high performance computing [32], key-value stores [30, 42, 43, 53], transactional processing systems [31, 67], and file systems [68]. These systems are largely complementary to GAIA.

1.5 Related Work by the PI

The PI is an expert on reliable datacenter software systems [27, 66] and reliable, secure multi-threading runtime systems [24–26, 70]. The PI's works are published in premier conferences on systems software (OSDI 2010, SOSP 2011, SOSP 2013, and SOSP 2015) and programming languages (PLDI 2012 and ASPLOS 2013). As preliminary results for this GAIA proposal, the PI has

developed a general consensus protocol [27] (part of **Objective 1**) and a fault-tolerant datacenter scheduler [66] (part of **Objective 2**).

2 Research Plan and Methodology

GAIA strenghens the reliability of datacenter computing with a holistic methodology. This section first proposes FALCON (§2.1), a fast and scalable consensus protocol, it then leverages FALCON to build a scheduler (§2.2) and a VM replication infrastructure (§2.3) to improve the availability of applications running in these infrastructures. Finally, this section describes research plan (§2.4).

2.1 Objective 1: Building Fast, Scalable Consensus via RDMA

This section describes a performance problem (§2.1.1) in existing PAXOS procotols and presents FALCON (§2.1.2), a fast, scalable PAXOS protocol by leveraging RDMA.

2.1.1 Problem: Consensus latency of existing PAXOS protocols scale poorly

Despite the wide deployments of PAXOS (§1.1), its high consensus latency makes many software systems suffer. For efficiency, PAXOS typically assigns one replica as the leader to invoke consensus requests, and the other replicas as backups to simply agree on requests. To agree on an input, at least one message round-trip is required between the leader and a backup. A round-trip causes big latency (hundreds of μs) as it goes through TCP/IP layers such as software network stack and OS kernel. This latency may be acceptable for leader election [11, 20] or heavyweight transactions [27, 45], but undesirable for key-value store applications [51, 61].

As replica group size increases, PAXOS consensus latency often increases drastically [33] due to the linearly increasing number of consensus messages. One common approach to improve PAXOS scalability is leveraging parallel techniques such as multithreading [11, 18] or asynchronous IO [27, 60]. However, the high TCP/IP round-trip latency still exists, and synchronizations in these techniques frequently invoke expensive OS events such as context switches.

Our preliminary study (Figure 3) ran four PAXOS-like protocols [11, 18, 27, 60] on 40Gbps network with only one client sending consensus requests. We found that, when replica group size increased from 3 to 9, the consensus latency of three protocols increased by 30.3% to 156.8%, and 36.5% to 63.7% of the increase was in OS kernel.

As the increasing popularity of RDMA, it becomes a promising approach to address the PAXOS performance problem. However, fully exploiting RDMA speed in software systems is widely considered challenging by the community [31, 43, 53, 59]. For instance, DARE [59] presents a two-round, RDMA-based PAXOS protocol in a sole-leader manner: leader does all RDMA workloads and backups do nothing. DARE was fast with 3$\sim$5 replicas. However, our study shows that, as replica group grows, DARE's sole-leader and two-round protocol incurred scalability bottlenecks: the consensus latency increased by 11.7x as the group growed by 35x (a linear increase, Figure 3).

2.1.2 Falcon: a fast, scalable RDMA-based PAXOS protocol

Our key observation is that we should carefully separate RDMA workloads among the leader and backups, especially in a scalability-sensitive context. Intuitively, we can let both leader and backups do RDMA writes directly on destination replicas' memory, and let all replicas poll their local memory to receive messages.

Although doing so will consume more CPU resources than a sole-leader protocol [59], it has three major benefits. First, the leader has less workloads. Second, both leader and backups participate in consensus, which makes it possible to reach consensus with only one round [50]. Third, all replicas can just receive consensus messages on their bare, local memory. An analogy is threads receiving other threads' data via bare memory, a fast and scalable computation pattern.

We present FALCON,[1] a new RDMA-based PAXOS protocol and its runtime system. In GAIA, all replicas directly write to destination replicas' memory and poll messages from local memory to receive messages, and our runtime system handles other technical challenges such as cheecking message delivery and recovering replica failures.

To support general, unmodified applications, FALCON stems from our preliminary general protocol, CRANE [27]. Within FALCON, an application runs as is. FALCON automatically deploys this program on replicas, intercepts inputs from an application's inbound socket calls (e.g., `recv`) with a Linux technique called LD_PRELOAD, and invokes its consensus protocol to enforce same application inputs across replicas. Figure 1 shows FALCON's architecture with four key components: our consensus protocol for enforcing same inputs across replicas, an output checker for checking network outputs across replicas with our same protocol, an in-memory consensus log, and a guard process that handles checkpointing and recovering application execute states.

Figure 2 shows FALCON's consensus protocol in normal case. The leader first executes the actual inbound socket call to get the actual inputs (**L1**, not shown), it then invokes consensus across replicas. All solid arrows (**L3** and **B2**) are ultra-fast RDMA direct writes to remote replicas' memory. All replicas poll from their own bare memory to receive consensus messages. To handle replica failures, all replicas persitently log inputs to local storage. For these writes, because the sending replicas only need to copy the sent data to local RDMA network interface card and then the writes finish on local site. FALCON is scalable when more backup replicas are added.

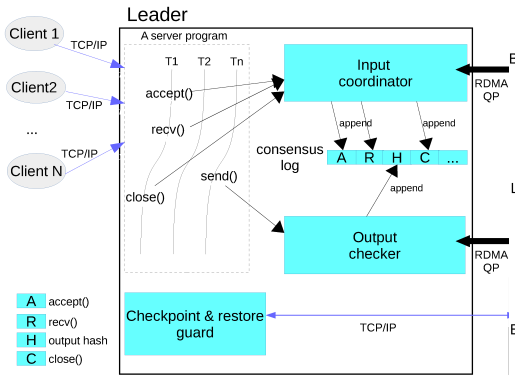

Figure 1: The FALCON architecture.

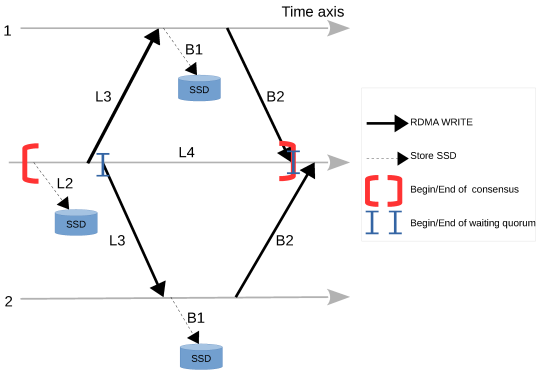

Figure 2: FALCON consensus protocol.

Primilinary results. Our preliminary results include two steps. First, to justify whether such a general, socket-intercepting protocol can support general applications, we have developed CRANE [27]. CRANE was able to support five widely used server applications (e.g., MySQL) without modifying them. Second, we built FALCON, a much faster and more scalable version of CRANE, by leveraging RDMA. Figure shoes FALCON performance with existing consensus protocols. FALCON was one order of magnitude faster latency than the literature. FALCON's consensus latency outperforms

[1] We name our protocol after falcon, one of the fastest birds.

4 popular PAXOS protocols by 32.3x to 85.8x on 3 to 9 replicas. FALCON is faster than DARE by up to 3.3x. When changing the replica group size from 3 to 105 (a 35x increase), FALCON's consensus latency increases merely from 8.2 μs to 31.6 μs (a 3.8x, sub-linear increase).

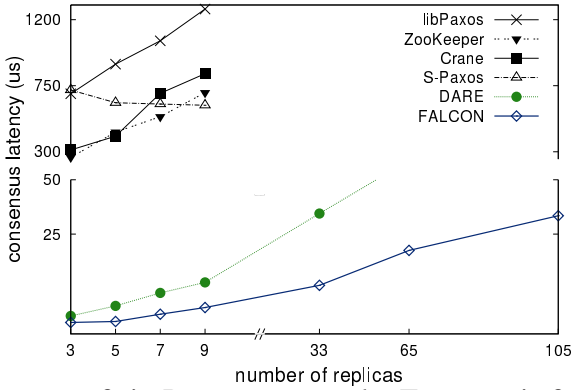

Figure 3: Consensus latency of six PAXOS protocols. FALCON is fastest and scales the best.

Future work. Since FALCON is the keystone of our objectives, this proposal plans to further study and improve its practicality in three aspects: (1) study its performance and potential bottlenecks on thousands of replicas, and propose new solutions; (2) study its protocol robustness on failure scenarios, including leader election and adding/removing replicas; and (3) evaluate its generality on more latency-critical applications.

2.2 Objective 2: Integrating FALCON with datacenter schedulers

Indeed, many existing schedulers have adopted PAXOS to improve availability for themselves. Typically, they use PAXOS to run multiple copies of their controllers, which works on scheduling computation jobs with resources. In normal case, only one leading controller does the real work, and the others standby to cope with the leader's failure. Unfortnately, most applications running by these schedulers have not been made highly-available (although minor applications implement a replication approach [15, 29]).

A naive approach to achieve high application availability could be implementing a PAXOS within each application. However, this approach has two major issues. First, PAXOS is notoriously difficult to understand [47, 56], implement [21, 50], or test [34, 69], thus developing a PAXOS protocol for each application is widely considered a nightmare [21, 34, 69] for application developers.

The second issue is, the scheduler may defeat PAXOS due to unawareness of the application's PAXOS replication logic. For instance, if an application submits multiple copies of the same computation job to the scheduler, the scheduler may incorrectly schedule several copies on the same computer (it should schedule each copy on different computers to achieve PAXOS fault-tolerance).

2.2.1 TRIPOD: the fault-tolerant scheduler architecture

This section proposes the design of TRIPOD, a scheduler infrastructure that automatically provides high-availability to general applications. TRIPOD is integrated with a widely used scheduler MESOS [38] and FALCON (**Objective 1**). To avoid the two aforementioned issues (§2.2), TRIPOD chooses to integrate PAXOS in a scheduler, not in applications. To achieve high application availability, unlike existing schedulers which let only one controller schedule jobs, TRIPOD runs

replicas of the same job using replicas of controllers: after controllers agree on a new job with FALCON, TRIPOD lets each controller independently schedule an copy of this job.

Doing so has three benefits. First, TRIPOD's PAXOS acts as a single, general fault-tolerance service to applications. We can just leverage existing verification tools [34, 69] to make sure that our FALCON protocol is robust and correct, and then we can benefit many applications. Second, application developers can now just focus on their own application logic, greatly saving development time and money. Third, now TRIPOD's own scheduler can handle the replication logic and do careful, replication-aware scheduling for jobs (§2.2.2).

Figure 4 depicts TRIPOD's architecture, and its key components are shaded (and in blue). To illustrate how TRIPOD works in an application perspective, this figure shows two applications, Hadoop and MPI. Each application has a *replica strength* ($\check{R}$) to denote the level of fault-tolerance it demands. This value is either 1 or equals the number of replicas of controllers in TRIPOD.

By default, each application has $\check{R}$=1, which means that this application does not need replication. For such a default setting, TRIPOD runs the job as is without replication, like a typical cluster management system (e.g., Mesos).

In this figure, Hadoop's $\check{R}$ is 3, which means that it wants to replicate each of its job with three copies for high-availability. Suppose Hadoop submits two jobs to the leader controller, each has different shapes (triangle or hexagon). The leader controller then invokes a consensus on each job across controllers. Once a consensus is reached, each controller assigns the same job on different slave machines. The leader controller directly returns its computation result to the Hadoop scheduler. Standby controllers ignore the results unless the same mechanism is triggered.

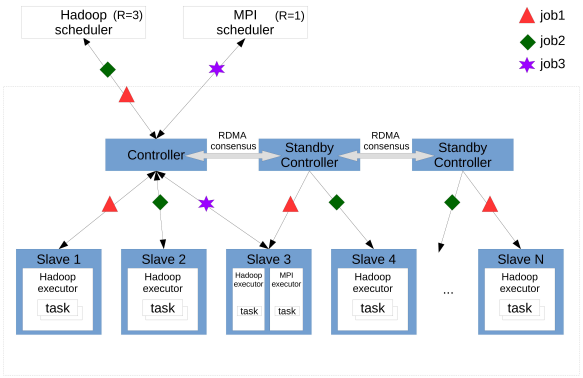

Figure 4: Fault-tolerant scheduler.

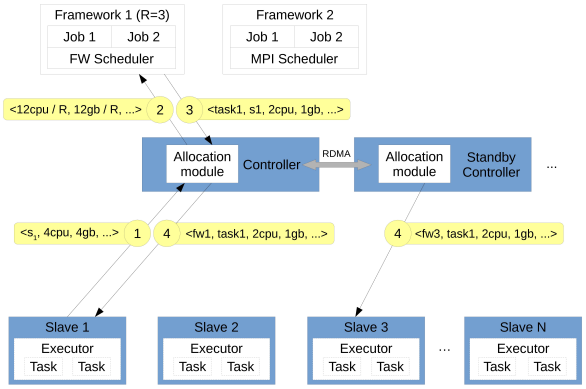

Figure 5: Workflow.

2.2.2 Replication-aware resource allocation workflow

Figure 5 shows TRIPOD's workflow on scheduling jobs with four steps. This workflow is similar to that in Mesos except the second and fourth steps. These two steps TRIPOD abstract away the replication logic in its resource offers and allocations from the application. An application runs as if GAIA does not replicate any of its jobs, and TRIPOD transparently handles all the replication logic.

In the first step, slave machines periodically report their available computing resources (e.g., CPU cores and memory) to the leader controller. In the second step, instead of offering the available resources aggregated from slave machines, TRIPOD divides the amount of resources by each application's $\check{R}$ value and then sends a resource offer to the application. The goal is to reserve enough resources for TRIPOD to replicate a job with $\check{R}$ copies.

In the third step, an application scheduler submits jobs to the leader controller. The leader controller then invokes a consensus on this job by carrying the resource offer made to the application.

Once a majority of controllers agrees on executing this job, each controller does the fourth step. It schedules this job on an available slave machine according to the resource offer. To prevent controllers putting the same job on the same slave machine, the leader controller first makes an assignment on which controller should run this job on which slave machine, it then carries this assignment in its consensus request. Once a consensus on this job is reached, each controller follows this assignment.

Availability v.s. resource consumption. TRIPOD is designed to make a mission-critical applicaion highly avaialble by leveraging Ř times of resources than the application's native, unreplicated execution. We deemed this extra resource consumption reasoanble for two reasons.

First, a major trend is that an application runs on more and more computers, thus minor computer failures tend to happen more likely. Such failures may turn down the entire application and cause if the failure computer runs a critical computation. For instance, Both NYSE and Nasdaq have experienced outage of their whole site [9] or specific IPO events [4] due to minor machine errors. In addition, social-networking applications like Facebook has strong fault-tolerance requirements, because minor machine failures have turned down the whole Facebook site for several times in the last few years [5], costing huge money lost.

Second, it is already a common practice to replicate critical computations by using Ř times of resources, and doing so can improve both availability and performance. For instance, Scatter [33] runs 8$\sim$12 replicas in a PAXOS group to order client requests, and it lets replicas reply requests in parallel. A bigger group size will improve Scatter throughput. Moreover, several recent replication systems [27, 35, 45] improve the availability of general server applications (e.g., MySQL [12]) by replicating them.

Primilinary results. We built a preliminary TRIPOD propotype, published in [APSys '16]. To evaluate a typical social-networking application, we ran TRIPOD with Memcached [51], a popular key-value store used by Twitter and financial platforms [37]. Compared to Memcached's unreplicated execution, TRIPOD incurred merely a 3.22% overhead in throughput and 3.31% in response time.

Future work. Our TRIPOD development will go along two directions. First, currently our replication and resource allocation workflow is tied with MESOS. We will study other popular schedulers and summarize their resource allocation workflow patterns, and we will develop a general, scheduler-agnostic workflow. Second, we will study new differential replication schemes, so that we can flexibly assign different Ř values to different compnents of an appilcation, getting both satisfiable availability and optimal resource consumption.

2.3 Objective 3: Strenghening VM to improve application availability

Virtual machines (VM) infrastructures (e.g., Amazon EC2 [14] and OpenStack [57]) are widely deployed in datacenters and clouds because they can provide a virtualized abstraction of computing resources to different applications and enforce strong utiilization isolation and security.

As mentioned in related work (§1.4), two approaches, primary-backup and live migration, exist for improving VM fault-tolerance, resource utilization, and energy saving. Both these approaches face problems on substantial application down time (e.g., 8 seconds in a live migration system vMotion [55]) and network bandwidth. The downtime hurts application availability even if there

is no replica failure. The bandwidth consumption often aggravates resource burdens because these approaches are often invoked when resources are tight.

A main reason that causes these two problems is that these approaches have only one actual execution of an application. Therefore, once execution states is required to transfered to a remote computer, the local execution has to be disturbed.

2.3.1 Forming an eco-system with VM and FALCON

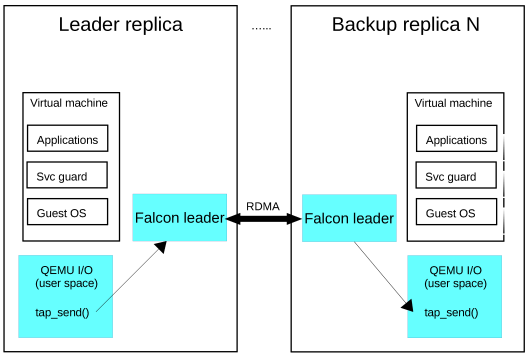

Figure 6: Performance overhead of scheduler. Our key components are shaded (and in blue color).

Fortunately, PAXOS-based replication can construct multiple, equivalent executions for the same application, thanking to its robustness and consistency. To ensure high application avaialbility, we can just run PAXOS to make replicas of VMs see the same sequence of inputs, and doing so is feasible because many VM architectures have input interception layers by default.

We make FALCON and VM form a mutual-beneficial eco-system. In this eco-system, VM provides a hypervisor layer to automatically capture incoming inputs and application execution state changes for FALCON, and FALCON benefits a VM by: (1) improving avaialbility of applications running in this VM, and (2) greatly improving application downtime and bandwidth consumption for the VM's live migration.

Figure 6 shows the architecture of the eco-system. To provide the same fault-tolerance guarantee with primary-backup, a typical replication factor of this eco-system is Ř. This eco-system chooses KVM [46] due to two main reasons. First, KVM is an open source hypervisor carried in Linux. Second, it provides t', an input capturing API at its QEMU component running at user space. Compared to other types of hypervisors, this API enables FALCON to coordinate inputs with RDMA, because currently RDMA only supports user space.

A key benefit of such a PAXOS-based VM replication over primary-backup is that its leadership is strongly consistent (through a majority aggrement, which primary-backup lacks), and it saves the bandwidth consumption for transfering application execution states in primary-backup.

2.3.2 PAXOS-based Live Migration

Interestingly, this replication ability not only provides high availability for fault-tolerance (suitable for the primary-backup scenario), but can also greatly save resource consumption if fault-tolerence is not a major concern (suitable for the live migration scenario). Consider the live migration scenario, PAXOS backups can only agree on inputs without actually executing them, then the overall application execution consumes almost same resources as the unreplicated one. A backup now can

just absorb occasional, periodical application checkpoints from the leader when the leader replica is idle, and catches up with the leader when a migration destination is decided on this backup.

Leveraging this idea, we propose a fast, network bandwidth-friendly live migration approach called "PAXOS-based live migration". A key benefit of this approach is that, now migration does not require transfering all execution states of application, but only tranferfing the PAXOS leadership (almost as fast as PAXOS consensus latency) to a destination backup which has idle resources. To increase the chance on finding a proper destination, we can leverage FALCON's high availability to run many backups within a datacenter.

This PAXOS-based live migration approach contains four steps. First, in normal case, the current computing instance (the leader) just agrees on incoming inputs with backups using FALCON. A set of backups run on hightly loaded computers and agree on inputs (also record the inputs persistantly).

Second, the leader does a periodical checkpoint when it's idle (no incoming requests), and sends the checkpoint to the backups. The backups only keeps the latest checkpoint and discards prior ones.

Third, when the VM invokes a migration on the leader computer, it uses a standard migration mechanism to determine which backup should be the migration destination (the next leader). Then, the destination extracts the lastest checkpoint on local computer, restores the application state, and then catches up with the leader with executing the recorded but not executed inputs.

Fourth, we invoke a FALCON protocol level operation: migration the leadership, which is similar to a traditional PAXOS leader election process [50], but we just explicitly decide the destination computer as the new leader. FALCON's protocol handles various failure scenarios (e.g., the destination computer crashes during the migration) as its PAXOS nature.

Future work. We plan to fully implement the eco-system, including both the replication architecture (§2.3.1) and the novel live migration approach (§2.3.2). We will compare the performance overhead of our replication approach with existing open-source VM infrastructure (e.g., OpenStack). We will also compare the performance of our migration approach with existing live migration approaches (e.g., vMotion).

2.4 Research Plan

This project will require two PhD students S1 and S2 to work for three years. In the first year, S1 will design and fully implement the FALCON protocol (part of **Objective 1**), and S2 will evaluate its performance and robustness on various real-world storage applications (part of **Objective 2**). In the second year, S1 will integrate FALCON to a scheduler MESOS (part of **Objective 2**), and S2 will make FALCON and KVM form an eco-system (part of **Objective 3**). In the third year, S1 and S2 will respectively study the efficacy of their systems built Object 2 and Object 3 with real-world applications, including big data applications.

References

[1] Why the data center needs an operating system. http://radar.oreilly.com/2014/12/why-the-data-center-needs-an-operating-system.html.

[2] Docker. http://http://criu.org/Docker.

[3] Data Plane Development Kit (DPDK). http://dpdk.org/.

[4] Facebook ipo: What went wrong? http://money.cnn.com/2012/05/23/technology/facebook-ipo-what-went-wrong/, .

[5] Is facebook down? a history of outages. https://www.theguardian.com/technology/2015/jan/27/is-facebook-down-outages, .

[6] An Introduction to the InfiniBand Architecture. http://buyya.com/superstorage/chap42.pdf.

[7] RDMA iWARP. http://www.chelsio.com/nic/rdma-iwarp/.

[8] LXC. https://linuxcontainers.org/.

[9] This is why the nyse shut down today. http://fortune.com/2015/07/08/nyse-halt/.

[10] Mellanox Products: RDMA over Converged Ethernet (RoCE). http://www.mellanox.com/page/products_dyn?product_family=79.

[11] ZooKeeper. https://zookeeper.apache.org/.

[12] MySQL Database. http://www.mysql.com/, 2014.

[13] Tupperware. https://www.youtube.com/watch?v=C_WuUgTqgOc, 2014.

[14] Amazon Virtual Private Cloud (VPC). https://aws.amazon.com/vpc/.

[15] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Effective straggler mitigation: Attack of the clones. In *NSDI'13*, 2013.

[16] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, 2003.

[17] C. E. Bezerra, F. Pedone, and R. V. Renesse. Scalable state-machine replication. In *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '14, 2014.

[18] M. Biely, Z. Milosevic, N. Santos, and A. Schiper. S-paxos: Offloading the leader for high throughput state machine replication. In *Proceedings of the 2012 IEEE 31st Symposium on Reliable Distributed Systems*, SRDS '12, 2012.

[19] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 285–300, Berkeley, CA, USA, 2014. USENIX Association. ISBN 978-1-931971-16-4. URL http://dl.acm.org/citation.cfm?id=2685048.2685071.

[20] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the Seventh Symposium on Operating Systems Design and Implementation (OSDI '06)*, pages 335–350, 2006.

[21] T. D. Chandra, R. Griesemer, and J. Redstone. Paxos made live: An engineering perspective. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing (PODC '07)*, Aug. 2007.

[22] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, 2005.

[23] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally-distributed database. In *Proceedings of the 12th Symposium on Operating Systems Design and Implementation (OSDI '16)*, Oct. 2012.

[24] H. Cui, J. Wu, C.-C. Tsai, and J. Yang. Stable deterministic multithreading through schedule memoization. In *Proceedings of the Ninth Symposium on Operating Systems Design and Implementation (OSDI '10)*, Oct. 2010.

[25] H. Cui, J. Wu, J. Gallagher, H. Guo, and J. Yang. Efficient deterministic multithreading through schedule relaxation. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP '11)*, pages 337–351, Oct. 2011.

[26] H. Cui, J. Simsa, Y.-H. Lin, H. Li, B. Blum, X. Xu, J. Yang, G. A. Gibson, and R. E. Bryant. Parrot: a practical runtime for deterministic, stable, and reliable threads. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP '13)*, Nov. 2013.

[27] H. Cui, R. Gu, C. Liu, and J. Yang. Paxos made transparent. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP '15)*, Oct. 2015.

[28] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174. San Francisco, 2008.

[29] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 10–10, 2004.

[30] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. Farm: Fast remote memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, 2014.

[31] A. Dragojević, D. Narayanan, E. B. Nightingale, M. Renzelmann, A. Shamis, A. Badam, and M. Castro. No compromises: Distributed transactions with consistency, availability, and performance. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP '15)*, Oct. 2015.

[32] M. P. I. Forum. Open mpi: Open source high performance computing, Sept. 2009.

[33] L. Glendenning, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Scalable consistency in scatter. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP '11)*, Oct. 2011.

[34] H. Guo, M. Wu, L. Zhou, G. Hu, J. Yang, and L. Zhang. Practical software model checking via dynamic interface reduction. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP '11)*, pages 265–278, Oct. 2011.

[35] Z. Guo, C. Hong, M. Yang, D. Zhou, L. Zhou, and L. Zhuang. Rex: Replication at the speed of multi-core. In *Proceedings of the 2014 ACM European Conference on Computer Systems (EUROSYS '14)*, page 11. ACM, 2014.

[36] Hadoop. Hadoop. `http://hadoop.apache.org/core/`.

[37] R. Hecht and S. Jablonski. Nosql evaluation: A use case oriented survey. *2012 International Conference on Cloud and Service Computing*, 0:336–341, 2011. doi: http://doi.ieeecomputersociety.org/10.1109/CSC.2011.6138544.

[38] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX conference on Networked Systems Design and Implementation*, NSDI'11, Berkeley, CA, USA, 2011. USENIX Association.

[39] M. Isard. Autopilot: Automatic data center management. *SIGOPS Oper. Syst. Rev.*, 41(2):60–67, Apr. 2007. ISSN 0163-5980. doi: 10.1145/1243418.1243426. URL `http://doi.acm.org/10.1145/1243418.1243426`.

[40] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys '07: Proceedings of the Second ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 59–72, 2007.

[41] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 261–276, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-752-3. doi: 10.1145/1629575.1629601. URL `http://doi.acm.org/10.1145/1629575.1629601`.

[42] J. Jose, H. Subramoni, K. Kandalla, M. Wasi-ur Rahman, H. Wang, S. Narravula, and D. K. Panda. Scalable memcached design for infiniband clusters using hybrid transports. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, 2012.

[43] A. Kalia, M. Kaminsky, and D. G. Andersen. Using rdma efficiently for key-value services. Aug. 2014.

[44] M. Kapritsos and F. P. Junqueira. Scalable agreement: Toward ordering as a service. In *Proceedings of the Sixth International Conference on Hot Topics in System Dependability*, HotDep'10, 2010.

[45] M. Kapritsos, Y. Wang, V. Quema, A. Clement, L. Alvisi, M. Dahlin, et al. All about eve: Execute-verify replication for multi-core servers. In *Proceedings of the Tenth Symposium on Operating Systems Design and Implementation (OSDI '12)*, volume 12, pages 237–250, 2012.

[46] Kernel Virtual Machine (KVM). `http://www.linux-kvm.org/`.

[47] L. Lamport. Paxos made simple. `http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf`.

[48] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[49] Y. Mao, F. P. Junqueira, and K. Marzullo. Mencius: building efficient replicated state machines for wans. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, volume 8, pages 369–384, 2008.

[50] D. Mazieres. Paxos made practical. Technical report, Technical report, 2007. http://www. scs. stanford. edu/dm/home/papers, 2007.

[51] Memcached. `https://memcached.org/`.

[52] E. Michael. *Scaling Leader-Based Protocols for State Machine Replication*. PhD thesis, University of Texas at Austin, 2015.

[53] C. Mitchell, Y. Geng, and J. Li. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *Proceedings of the USENIX Annual Technical Conference (USENIX '14)*, June 2013.

[54] I. Moraru, D. G. Andersen, and M. Kaminsky. There is more consensus in egalitarian parliaments. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP '91)*, Nov. 2013.

[55] M. Nelson, B.-H. Lim, and G. Hutchins. Fast transparent migration for virtual machines. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, 2005.

[56] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the USENIX Annual Technical Conference (USENIX '14)*, June 2014.

[57] OpenStack: Open Source Cloud Computing Software. https://www.openstack.org/.

[58] S. Peter, J. Li, I. Zhang, D. R. K. Ports, D. Woos, A. Krishnamurthy, T. Anderson, and T. Roscoe. Arrakis: The operating system is the control plane. In *Proceedings of the Eleventh Symposium on Operating Systems Design and Implementation (OSDI '14)*, Oct. 2014.

[59] M. Poke and T. Hoefler. Dare: High-performance state machine replication on rdma networks. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, 2015.

[60] M. Primi. LibPaxos. http://libpaxos.sourceforge.net/.

[61] Redis. http://redis.io/.

[62] R. Van Renesse and D. Altinbuken. Paxos made moderately complex. *ACM Computing Surveys (CSUR)*, 47(3):42:1–42:36, 2015.

[63] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2428-1. doi: 10.1145/2523616.2523633. URL http://doi.acm.org/10.1145/2523616.2523633.

[64] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, pages 18:1–18:17, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3238-5. doi: 10.1145/2741948.2741964. URL http://doi.acm.org/10.1145/2741948.2741964.

[65] C. A. Waldspurger. Memory resource management in VMware ESX server. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, 2002.

[66] C. Wang, J. Yang, N. Yi, and H. Cui. Tripod: An efficient, highly-available cluster management system. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '16, 2016.

[67] X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen. Fast in-memory transaction processing using rdma and htm. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP '15)*, SOSP '15, Oct. 2015.

[68] G. G. Wittawat Tantisiriroj. Network file system (nfs) in high performance networks. Technical Report CMU-PDLSVD08-02, Carnegie Mellon University, Jan. 2008.

[69] J. Yang, T. Chen, M. Wu, Z. Xu, X. Liu, H. Lin, M. Yang, F. Long, L. Zhang, and L. Zhou. MODIST: Transparent model checking of unmodified distributed systems. In *Proceedings of the Sixth Symposium on Networked Systems Design and Implementation (NSDI '09)*, pages 213–228, Apr. 2009.

[70] J. Yang, H. Cui, J. Wu, Y. Tang, and G. Hu. Determinism is not enough: Making parallel programs reliable with stable multithreading. *Communications of the ACM*, 2014.

[71] M. Zaharia, B. Hindman, A. Konwinski, A. Ghodsi, A. D. Joesph, R. Katz, S. Shenker, and I. Stoica. The datacenter needs an operating system. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*, 2011.

[72] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, and J. Xu. Fuxi: A fault-tolerant resource management and job scheduling system at internet scale. *Proc. VLDB Endow.*, 7(13):1393–1404, Aug. 2014. ISSN 2150-8097. doi: 10.14778/2733004.2733012. URL http://dx.doi.org/10.14778/2733004.2733012.