The Tortoise and the Hare: Characterizing synchrony in distributed environments (Practical experience report)

Daniel Porto*, João Leitão†, Flavio Junqueira‡, Rodrigo Rodrigues*
daniel@gsd.inesc-id.pt, jc.leitao@fct.unl.pt, fpj@apache.org, rodrigo.rodrigues@inesc-id.pt
INESC-ID / Instituto Superior Técnico, ULisboa, NOVA LINCS / FCT - UNL†, Dell EMC‡*

***Abstract*—The design of distributed protocols that run in data centers and enterprise clusters is heavily dependent on synchrony assumptions regarding the timing behavior of the participating nodes and the network. However, little is known about the actual synchrony of real distributed systems, and how it varies across deployments. To better understand this timing behavior and how it impacts the design and implementation of distributed protocols, we conduct an extensive measurement study of the latency for transmitting and processing messages between nodes in four different environments. Our study determines how protocol characteristics affect the latency behavior. We also determine how different environmental factors can affect the measured latency and whether high latency events manifest globally or locally. Our results suggest several directions for reducing latency, and for leveraging recent distributed computing models in a more judicious way.**

I. Introduction

Distributed protocols are at the foundation of the systems that form the infrastructure of today's computing services. Examples of such systems range from the ecosystem of distributed systems that run in data centers for supporting cloud services, to smaller scale parallel or replicated systems that run in some enterprise clusters.

The design and the dependability properties of distributed protocols is crucially dependent on timing assumptions regarding the behavior of participating nodes and the network. In the design of distributed algorithms, these timing assumptions are encapsulated in a synchrony model (e.g., synchronous, asynchronous, partially synchronous, etc.)

Nowadays, many deployed systems are based on algorithms, such as Paxos, that do not require any synchrony assumptions for safety [1]–[3]. Such an assumption of an asynchronous system is conservative, but clearly safe. Conversely, assuming a synchronous system enables protocols with fewer replicas, e.g., one can solve consensus with only $f + 1$ processes [4]. However, guaranteeing that the timing assumptions of a synchronous system are not violated in current data centers is very difficult without very long time bounds, which leads to the typical conservative choice of modeling the system as asynchronous.

Given the general trend towards an increased predictability of data center systems [5] and networks [6], [7], it is possible that the calibration of such timers will become easier and encourage designers of dependable distributed systems to make more optimistic assumptions [8], [9]. However, it is important to understand whether this optimism is warranted, and what factors may break such predictability.

In this paper, we shed some light on these questions by conducting an extensive measurement study of the communication latency between participating nodes of a distributed system in four environments, corresponding to different modern and very distinct deployment scenarios: two public cloud IaaS environments, and two private environments with demanding workloads, namely a small cluster operated by a software company for testing and an academic cluster for conducting research experiments.

In our study, we not only continuously measure the latency between processes running on different machines in each of these environments, but also try to understand how sensitive is latency to protocol characteristics such as the size of the protocol messages, the type of processing involved, or the need to access the disk as part of the processing of protocol messages. Additionally, we determine how some environmental factors affect the measured latency. These factors include CPU, memory, and disk usage, garbage collection (GC) activity of the processes involved, or network traffic. Finally, we also study how different participants perceive the latency towards other processes.

Our main findings are summarized as follows. (1) We found that while these environments lead to mostly low latency values, the latency distribution also exhibits a long tail, which is accentuated in scenarios with heavy co-located workloads. (2) Some characteristics of the distributed protocols affect the observed latency more significantly than others (e.g., one of the most penalizing aspects is logging information to disk synchronously before issuing a reply). (3) High communication latency is normally a characteristic of nodes, and not the network, and therefore it is perceived globally in an identical way across all the participants of a distributed protocol. (4) From the environmental factors that affect latency, the one with the strongest positive correlation is the amount of time the node spends in GC activity.

In the remainder of the paper, after describing our deployments (§II) and methodology (§III), we present the analysis of the collected data (§IV). Then we discuss (§V) how these findings: (1) allow for several interesting avenues for optimizing the latency in similar environments, and (2) lead to a much more informed and judicious way to parameterize the emerging class of system models for building dependable

systems that leverage the increased predictability of enterprise and data center environments [8], [9]. We conclude with a survey of related work (§VI) and final remarks (§VII).

II. Target systems

Infrastructure systems can range from simple deployments with a few nodes (e.g., simple Web applications backed by a database) to deployments spanning many clusters and data centers. The behavior of each individual system depends on a number of factors, including the amount and complexity of processing it performs, the compute resources it uses, and the interference of coexisting systems. As a first example, consider the case of search engines, where a query is scattered across a number of search nodes to parallelize the processing; individual nodes process a partition of the document collection to obtain candidate results. Such complexity can induce some imbalance in the processing of a query, despite attempts to balance the computation across nodes. Search nodes heavily use memory resources and access the disk occasionally. They can also share resources with other systems [10], [11]. Data analytics frameworks similarly present complex processing and the presence of stragglers has been the subject of extensive work [12]. As a final example, consider coordination services like Chubby [2] or ZooKeeper [3]. In contrast with the other examples, these have very lightweight processing and the complexity is in the protocol itself. Such services access disk for durability and memory heavily, and are typically latency sensitive. Understanding the latency behavior of such systems entails a precise characterization of the protocol behavior.

To be able to derive more general observations despite this diversity of infrastructure systems, we focus on the Remote Procedure Call (RPC) primitive that many systems rely upon rather than focusing on a specific application or class of applications. We investigate in this work the behavior of RPCs between pairs of processes and from an initiating process to a group of other processes. Such a scatter-gather pattern of group RPCs is present in a number of systems: search engines often have a master that sends a query to a group of processes and collects the results; in consensus protocols, a leader broadcasts a propose and collects acknowledgments; in distributed log engines, such as Apache BookKeeper, a client broadcasts a record write and collects acknowledgments; systems like Apache Kafka use a variant of such a group RPC where replicas pull updates from a leader instead of having the leader initiating the replication.

We focus only on the latency of RPCs for simple procedure implementations. In particular, we implement both group and point-to-point RPCs, and we vary the RPC implementation to exercise CPU, network, and disk IO. We acknowledge that the effects of complex and unbalanced processing can further affect the latency observed in infrastructure systems, but understanding such application-specific effects is beyond the scope of our study.

Table I: Overview of data collected

Data	Pub. Cloud 1	Pub. Cloud 2	Enterprise	Academic
Period (days)	6	4	23	219
Samples RPC	6M	2M	16M	261M
Samples RPC + status	-	97%	15%	40%
Group sizes	3,5,7	4,7	3,4,7	3,4,5,7,9

III. Data sets

We conducted our study in following environments.

Public Cloud 1 and Public Cloud 2. These are two popular IaaS platforms, i.e., they offer the ability to rent virtual machines (VMs) on demand. We co-located all processes within the same data center, but we have no information whether they are co-located at the same server.

Academic cluster. This is a cluster of a research institute with over 60 researchers. The cluster has around 70 machines with 2.67GHz Intel Xeon CPUs, 48GB of RAM, bonded Ethernet interfaces, each with 1 Gbps. The OS is a Debian Linux server edition without virtualization. All machines are connected to two interconnected 1Gbps switches. Researchers use the cluster for their experiments, which make heavy use of cluster resources, namely CPU.

Entreprise cluster. This is a cluster at a software development company with hundreds of employees. Its main product is a platform for the rapid prototyping and management of enterprise web applications. This is a very large software platform, which is constantly evolving. This cluster is mostly used for compilation and testing of this software. It is composed of seven physical machines each with an Intel Xeon dual core CPU (2.20GHz) and 192GB of RAM. Machines are connected through a set of switches organized in a tree. This cluster is virtualized using VMWare VSphere.

For each of the above environments, we deployed a set of active agents that measure latency among themselves, and a set of passive agents that continuously monitor the environment on which the experiments run. Both the active and passive agents were implemented in Java. The active agents use the Netty library (v3.6.2) to support communication among different processes. The passive agent was designed to have minimal impact on the performance of other processes running in the system, following the guidelines presented in [13]. The active agents operate in an open loop, and periodically (every second) issue an RPC to a subset of other active agents, and then measure the elapsed Round Trip Time (RTT). To capture the features that vary across distributed protocols, we started by partitioning the active agents into groups (in a similar fashion to how a stateful service partitions its state into several replica groups), with the RTT measurements being conducted within all members of each group. Participants of a group are changed daily. We deployed groups of size varying from 3 to 9 as presented in Table I. Thus, each agent issues periodically a "group RPC", consisting of sending an RPC request to all other members of its group (including itself), waiting for a reply from all of them, and measuring the arrival time of each reply. In addition, the active agent also stores statistics

Table II: Passive agents: resource usage metrics

Resource	Metrics
CPU	% of aggregated usage of all cores
Disk	IO queue size and time writing
Memory	used/swap, page faults, stalls, #swap page IO
Network	usage, errors, packet drops and collisions
Process	Garbage collection duration and calls

of garbage collection (GC) calls. However, group size is not the only factor that influences the latency between the participants of distributed protocols. As such, we define five "workloads", corresponding to different RPC handlers that exercise different resources. Within a group, each agent rotates the workload it uses for each RPC in a round-robin fashion, in the following order:

A (null): a 28 Byte request / 44 Byte reply pair, carrying an identifier of the group RPC session and timestamp. The handler for this workload does not perform any computation.

B (crypto): Same as **A**, except that the RPC handler at the receiver side computes a cryptographic hash, to illustrate a protocol that requires some CPU cycles to handle requests.

C (disk): Same as **A**, except that the handler of the RPC at the receiver side will write the reply to disk, to illustrate a protocol that logs information to disk synchronously as part of handling protocol requests.

D (network): Same as **A**, plus the request/reply have a 50KB payload, to capture the effects of large protocol messages.

E (all): This puts together the union of the above features.

We also run in every node (i.e., server for the Academic cluster deployment and virtual machine instance for Public Cloud 1, Public Cloud 2, and Entreprise cluster) an additional passive agent, which is materialized by an independent process that periodically (every 1 second approximately) obtains and stores from the underlying operating system statistics concerning overall resource usage, namely: %CPU usage based on jiffies at all cores; memory usage; network utilization and failures; disk accesses for read/write operations. A detailed list of all monitored aspects can be found in Table II.

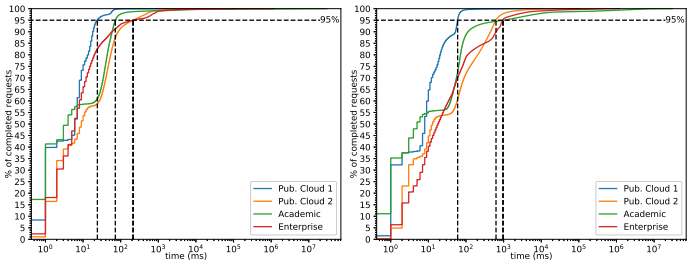
Finally, we summarize the key metrics we collected in Table I. The differences in measurement periods and group sizes are explained by cost constraints for public clouds and on the length of our access to the enterprise cluster.

IV. Analysis

This section presents and analyzes the results from our measurements. We split the analysis into four parts: an overview of the latency distributions, an analysis of its sensitivity to protocol design features, whether latency is a global or a local phenomenon, and its correlation with environmental factors.

We guide our analysis by the questions that each set of measurements helps answering, starting with the following.

Question 1. *What is the overall distribution of communication latency, and how does it vary across deployments?*



(a) RPC Latency - pairwise (b) RPC Latency - group

Figure 1: Overall latency distribution of completed requests

To analyze the overall distribution of inter-process communication latency in the different environments, we report the results for two kinds of RPC measurements: individual and group RPCs. Individual RPCs measure the time for a single caller and a single callee to communicate, while group RPCs have a single caller that sends the RPC to all members of the group and measures the time to collect the last answer from the entire group.

Figure 1a presents the CDFs for all the individual RPC latencies that we measured in the four deployments. Overall, the latencies observed at Public Cloud 1 are more predictable than in the other environments, since almost all measurements fall into a narrow interval of up to a few tens of milliseconds. When comparing the remaining three deployments, the latency measurements at Public Cloud 2 ramp up slower than the other two, but are faster than the other two deployments when comparing a small subset of the slowest RPCs (barely visible in Figure 1a, but more clear in Figure 1b). This shows that these two environments have longer tails of significant latency values than a public IaaS service.

Figure 1b shows the corresponding measurements for group RPCs. The behavior observed is similar to the one for the individual RPCs, except that the long tails became even longer. This is not surprising given that there are more processes being contacted per data point, which reflects the slowest RPC within the group. Interestingly, Public Cloud 1 still has a very short tail with over 95% of the group RPCs below 100 ms.

Next, we consider the effects of protocol characteristics. To this end we observe that, in distributed protocols, each process can be seen as a state machine that holds a set of protocol variables, and process incoming message by invoking the corresponding handler, which updates the state and optionally emits an output according to the protocol logic. The work of the handler varies significantly from protocol to protocol. For example, replication protocols that tolerate Byzantine faults use cryptographic primitives in their message handlers [14], whereas crash fault tolerant protocols do not [15]. Another relevant characteristic is that protocols often persist key information to a log on disk before issuing a reply to protocol messages [15]. However, not all protocols do this, namely those that require the constant availability

of a number of correct replicas that prevent such side effects from being lost [14]. Protocol message sizes can also vary significantly, for instance, depending on whether these messages carry any data (such as an application request or response) in their payload. To understand how these different protocol characteristics affect the latency between processes, we separately analyze the distribution of measured RTTs according to the characteristics of the RPC.

Question 2. *How does latency vary depending on the characteristics of the distributed protocol, such as group or message sizes, or type of processing for each protocol step?*

We start by analyzing in Figure 2 the effect of participating in groups of different size. This figure contains four plots, one per deployment, where each plot contains several CDF curves of individual RPC latencies, one per group size. The results indicate that the group size does not provide a visible influence on the latency between processes at Public Cloud 1, Public Cloud 2, and at the Academic cluster. This is somewhat expected as the extra work required from the initiating process is to send a few extra messages and process the respective replies, which is sufficiently small to have a negligible impact. Furthermore, in these two clusters, the smaller groups tend to observe slightly lower latency than the larger ones, as expected. For the Entreprise cluster, the differences are more pronounced and the direction is the opposite of what we expected: larger groups have a lower latency. We do not have a definitive explanation for this effect, though we hypothesize that larger groups have a higher probability of containing at least one pair of physically co-located virtual machines.

Next, we consider again all group sizes and vary the workload that is being used, in order to gain an understanding of the effect of the various message handlers on RPC latency. The results are shown in Figure 3.

The results show that adding only a cryptographic operation (workload B) has little impact on the overall RPC latency. This is because a cryptographic computation is a CPU intensive task that adds little time to the message transmission in the absence of a high CPU load. In contrast, the remaining features have a visible impact on latency. Furthermore, this impact depends on the deployment: including only disk writes (workload C) leads to a larger increase in latency than using only large messages (workload D) in Public Cloud 2 and the Academic cluster, but both lead to comparable (and visible) increases in latency in Public Cloud 1 and the Entreprise cluster. This could be explained by a larger discrepancy in some environments between the latency of writing to disk and the latency for transmitting and processing messages with tens of kilobytes. As expected, the combination of features (workload E) leads to the highest latency values. These results suggest clear avenues for the optimization of distributed protocols, namely by employing techniques for reducing the cost of logging, such as batching groups of operations and parallelizing storage and processing [16].

The next analysis addresses the following question.

Question 3. *How differently is latency perceived by the various processes in the same group?*

In particular, we would like to understand whether there is any discrepancy on how latency is perceived across processes of the same group at any given time, i.e., whether a slow process is simultaneously perceived as slow by all or just a subset of the members of the group.

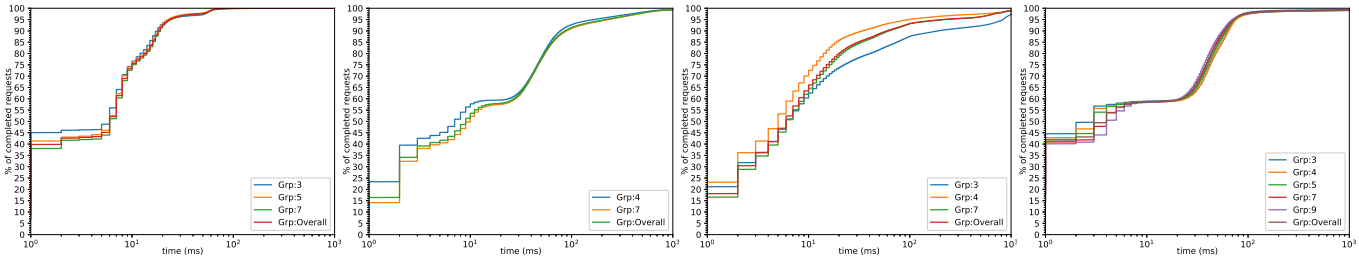
To understand this intra-group behavior, we had to conduct a separate analysis for each period of a single day because, during our data collection, we changed the composition of the groups after each period of 24 hours.

Through an initial observation of the data, we realized that several 24-hour periods manifested the pattern that is illustrated in Figure 4, which plots the measured latencies for one day of RPCs exchanged among processes within a group of 3 members at Public Cloud 1. Each of the first three plots (4a, 4b, and 4c) depicts the latency of RPCs that initiated at a fixed process (nodes 1, 8, and 14). We can see the same pattern irrespectively of the initiating process: a large fraction of about 97% of the messages sent to node 1 and node 8 are fast, namely arriving in at most 50ms, whereas a large fraction of 97% messages to node 14 require more than 80ms to arrive. Conversely, Figures 4d, 4e, and 4f group these latencies by the process receiving the RPCs. We can see that node 14 always has a long tail latency, even with regard to itself. The same pattern was observed in several occasions in the remaining clusters, except for Entreprise cluster where RPC latencies were overall evenly spread.

This pattern suggest that slowness is most likely a property of the processes and not of an external factor such as the network, since the slower processes are consistently perceived as slow even by themselves. Furthermore, this shows a very coherent view of slowness, where all the processes, including the slow one, perceive a very similar set of slow processes.

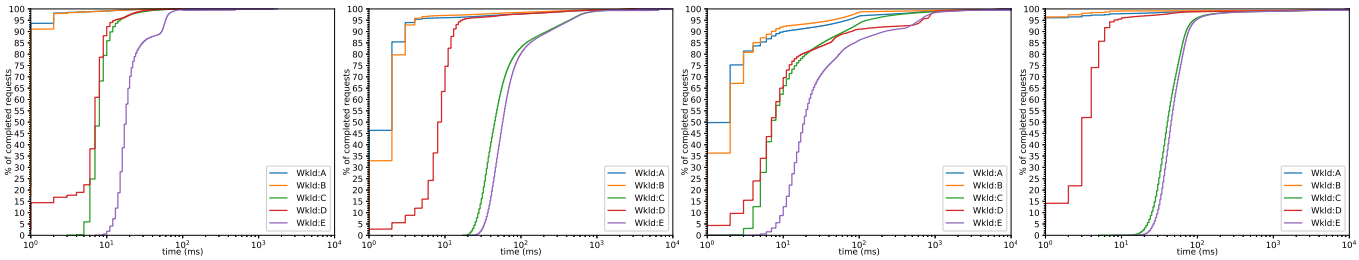
To understand whether we can generalize these conclusions, we need to characterize whether similar patterns hold for the remaining 24-hour periods of each deployment. To achieve this, we start by defining the notion of a "slow node", so that we can characterize the distribution of the number of slow nodes throughout the trace.

To find a precise way to capture the pattern of latencies from slow nodes in, e.g., Figure 4, we define that process x belonging to group G is perceived by y to be a slow node when the $90th$ percentile latency of RPCs from y to x is at least $2\times$ that of the RPCs from y to the fastest process (i.e., the process from G with the minimum $90th$ percentile latency from y). In figure 4a, for instance, it is clear that node 14 is perceived by node 1 as being slow since the $90th$ percentile latency of RPCs to 14 is significantly higher than the $90th$ percentile latency of RPCs from 1 to itself (the fastest one). (Note that the choice of $2\times$ and the $90th$



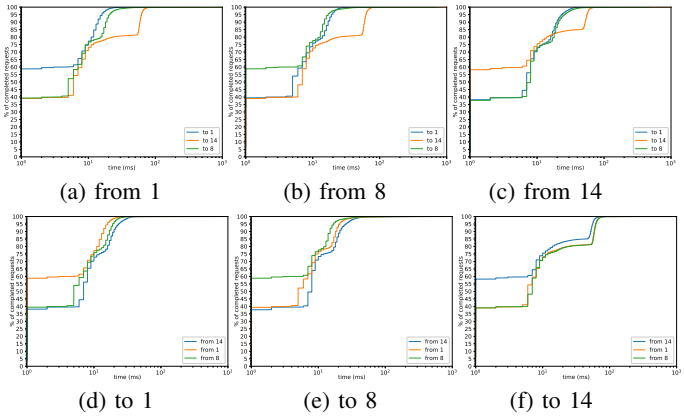
(a) Public Cloud 1 (b) Public Cloud 2 (c) Entreprise cluster (d) Academic cluster

Figure 2: Latency distribution for RPC measurements with respect to group size



(a) Public Cloud 1 (b) Public Cloud 2 (c) Entreprise cluster (d) Academic cluster

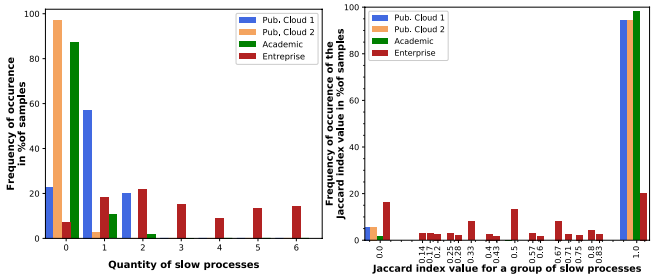
Figure 3: Latency distribution of each group with respect to each workload



(a) from 1 (b) from 8 (c) from 14

(d) to 1 (e) to 8 (f) to 14

Figure 4: One-day intra-group latencies for Pub. Cloud 1



(a) (b)

Figure 5: Histograms of number of slow processes per 24 hour period (a) and Jaccard index values (b).

percentile is arbitrary, and as future work we would like to understand whether varying this choice affects significantly these results.)

Based on this definition, we computed the distribution of slow nodes across all days of the study, from the perspective of all processes in groups of *size* $= 7$ (the largest group in most deployments). Figure 5a shows that there is a visible presence of slow processes in several days of the trace, with the Entreprise cluster being significantly more heterogeneous than the other deployments.

Furthermore, we wanted to confirm whether the pattern of the same process being perceived as slow by all processes (including itself) was common across the experiment. To achieve this, for each day where at least two processes x_i, x_j perceive a non-empty set of slow nodes S_i, S_j, we compute the Jaccard similarity coefficient of these sets, i.e., $J(S_i, S_j) = \frac{|S_i \cap S_j|}{|S_i \cup S_j|}$.

Figure 5b shows the distribution of this similarity coefficient for the days where more than one process perceives at least one other process as slow. The measured data shows that the similarity coefficient is very high (almost always 1 in three of the deployments), suggesting that slowness is a global phenomenon in most deployments, i.e., all processes perceive the same subset of processes as being slow.

Finally, we analyze the existence of correlations between high latency and various different conditions at the processes and the network.

Question 4. *How does the latency correlate with environmental factors, such as the usage of various resources at the host and the network?*

To answer this question, we paired each RPC sample with measurements concerning the usage of various resources at the host, as well as measurements concerning the usage of the network. Since these resource usage measurements were taken periodically, independent of the RPC activity, we

chose to pair each RPC latency value with the measurement that was conducted at the time instant that is closest to the moment the receiver recorded the RPC.

To analyze this data in search of correlations between latency and these various environmental factors, we present scatter plots of latency versus resource usage (only for a subset of influential resources), as well the result of computing the Pearson correlation coefficient, which is depicted in Figure 6. This shows that there is a strong correlation between latency and the amount of time spent by the GC, confirming that GC activity is likely to negatively affect the latency observed by distributed protocols. This result opens interesting avenues for research, e.g., on coordinating the GC activity of different processes in a distributed protocol (e.g., as in a holistic runtime system [17]) to confine the asynchronous behavior to a small subset of processes, which can then be exploited by the distributed computing model to decrease the replication factors [8], [9]. The results also show a moderate correlation of about 0.4 between latency and the usage of several resources, namely disk writes, page swapping (in and out), network usage and memory stalls.

The scatter plots in Figures 7 through 10 illustrate the correlations described previously. For instance, results indicate a strong positive correlation between latency and GC in Figure 8. These plots also allow for making other observations concerning resource usage. For example, the existence of horizontal lines in Figure 7 indicates that there is a strong prevalence of certain values for the CPU load. After analyzing these values, we noticed they correspond to a CPU utilization that is a multiple of $\frac{1}{\#cores}$, thus indicating a pattern where a subset of the CPUs might be saturated, while the remaining ones are idle. This stresses the need for writing concurrent code in order to fully utilize existing CPU resources.

V. Discussion

The insights gained from the analysis can improve the design of dependable distributed systems. For example, there are several distributed computing models that leverage stronger assumptions about timing to simplify the protocol and improve several of its aspects, such as message complexity, number of communication steps, or replication factors. E.g., assuming a synchronous model instead of asynchronous allows for solving consensus with only $f+1$ replicas instead of $2f+1$ [4]. However, if these stronger assumptions are not met, the system may violate its safety or liveness properties.

Timing assumptions are not "all or nothing" though, and several models allow for setting knobs that trade the strength of these assumptions for the improvements in the characteristics of the resulting protocols. Our findings may help calibrating those knobs in a way that maximizes such protocol improvements without incurring in violations of the specification of the system when those assumptions become unrealistic. We discuss three such instances.

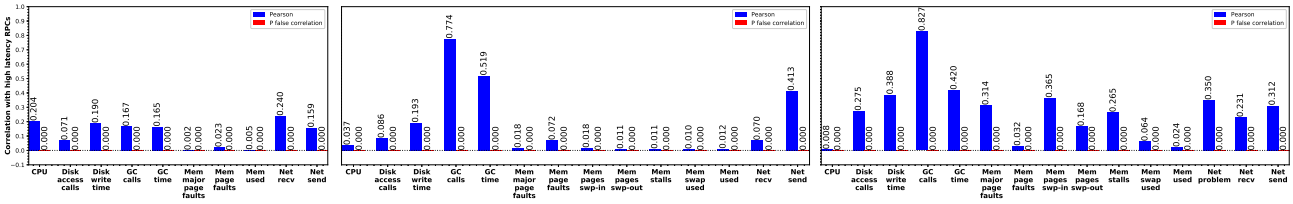
Γ-accurate failure detectors [18] allow for the output of a failure detector to meet a relaxed accuracy property that is limited to being accurate concerning a subset of the processes, Γ. The authors then show the degree of accuracy that is required to solve consensus in this model. In a crash fault-tolerant system, failure detection is done by means of timeouts. For instance, a failure detection module exchanges keep alive messages between processes and reports those that do not reply within a timeout as being suspected. In this setting, our findings can be used to determine how to set the cardinality of Γ as a function of the timeout value for different deployments. For instance, this cardinality can be set by determining, for the appropriate deployment scenario and protocol characteristics, a sweet spot in the curve that computes the likelihood of false positives as a function of the value of the timeout (which can be directly derived from our measurements).

Visigoth fault tolerance [8] is based on a system model where each processes perceives a bounded subset of s slow but correct processes, i.e., other system processes that do not obey a synchronous bound with respect to it (where this bound is set by a tunable parameter T for the time for inter-node communication). This model allows for decreasing the number of replicas required for consensus-based state machine replication from $2f+1$ to $f+s+1$. As such, there is a trade-off where the larger the value of T, the smaller s, and consequently the total number of replicas can be further cut, at the expense of higher timeouts and lower protocol performance. Again, our RPC latency measurements can be used to set the parameter s in an informed way. In particular, we can use our measurements to plot the value of s as a function of T, and find a sweet spot in the above trade-off.

The XFT model [9] provides correctness up to a threshold of a combination of faults and asynchrony. In short, the system can either tolerate several faults or several slow processes but not both. Just like in the Visigoth model this model uses a configurable timeout value that determines when a process is considered to be slow. As such, we could also use our findings to determine the likelihood of processes with an asynchronous behavior and thus contribute to the above threshold. Based on this likelihood, we obtain an informed estimate for how the actual tolerance of the system to crash faults will vary throughout the system lifetime.

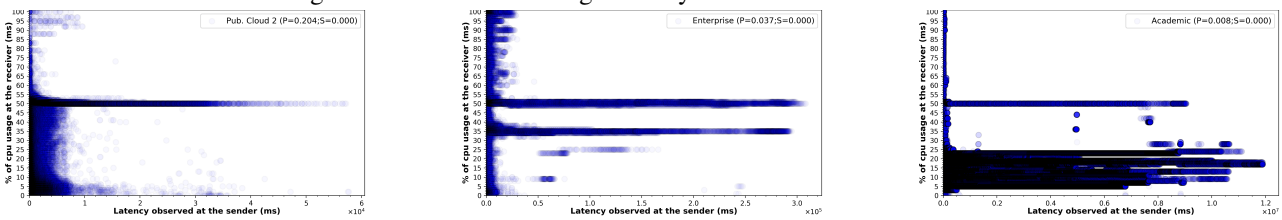
VI. Related work

The most closely related work are various measurement studies of the performance of cluster and data center systems. In particular, there exist several studies of the performance of a variety of cloud services [19]–[26], some of which also measured the distribution of RTT times between IaaS instances [19], [24], [26]. Our work has two main distinctions regarding these prior studies. First, we go beyond just measuring performance characteristic such as latency and throughput, namely by taking the point of view of the designer of distributed algorithms and trying



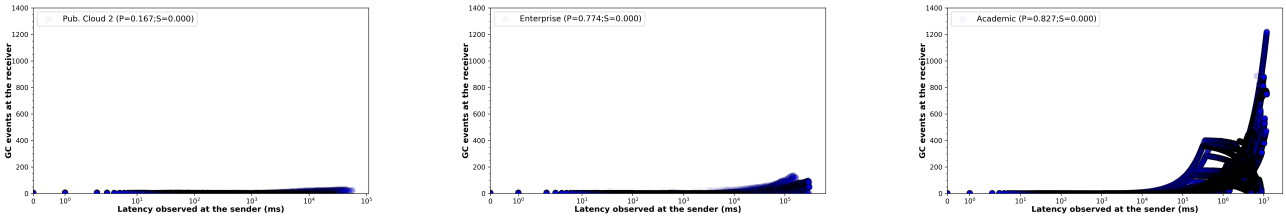
(a) Public Cloud 2 (b) Entreprise cluster (c) Academic cluster

Figure 6: Correlation of high latency RPCs and resource utilization



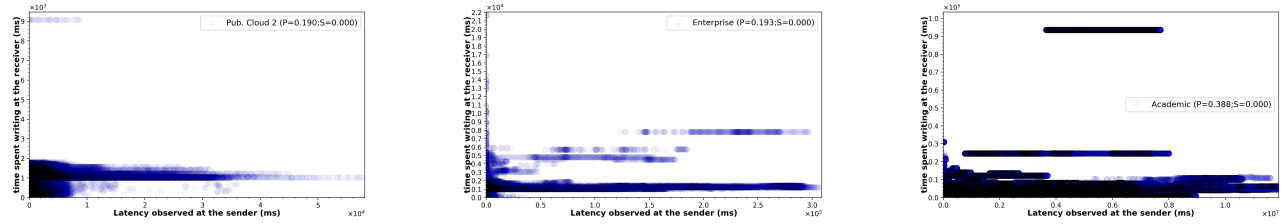
(a) Public Cloud 2 (b) Entreprise cluster (c) Academic cluster

Figure 7: CPU usage as a function of the latency of individual RPCs



(a) Public Cloud 2 (b) Entreprise cluster (c) Academic cluster

Figure 8: Garbage collection calls as a function of the latency (log scale) of individual RPCs



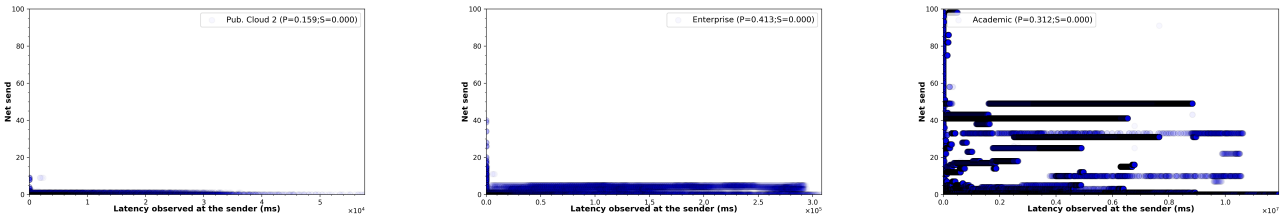
(a) Public Cloud 2 (b) Entreprise cluster (c) Academic cluster

Figure 9: Number of disk writes as a function of the latency of individual RPCs

to understand how protocol features influence the timing assumptions and distributed computing model. Second, we compare the timing characteristics of IaaS with an enterprise cluster and an academic cluster, and we correlate those measurements with data regarding the environment, which allows us to understand how these assumptions may need to be adapted according to the deployment characteristics.

Another set of studies measured the characteristics of data center networks [27]–[29]. The focus of these studies is even more distant from ours, since their emphasis is on the network characteristics. In particular, they measure aspects such as link utilization, number of active flows, flow and packet size and interarrival times, packet drop statistics, etc.

In the distributed computing community, a series of papers proposed system models that make different timing assumptions regarding the ability for two processes in a distributed system to communicate. For example, Dwork, Lynch, and Stockmeyer have proposed models with an unknown bound for message delays or with a known bound that is only guaranteed after some unknown time [30]. Such bounds can be expressed with failure detector abstractions [31]. Cristian and Fetzer have proposed the timed-asynchronous model where processes have bounded clock drifts [32]. Veríssimo and Casimiro proposed the Timely Computing Base model, where a synchronous and an asynchronous subsystem can coexist in the same execution [33]. Guerraoui and Schiper have proposed Γ-accurate detectors, where timeouts are only an accurate way to infer liveness with respect to a subset of the processes [18]. Both the VFT [8] and XFT [9] models, mentioned previously, assume that a subset of the system nodes observe timing bounds. Our work is complementary to these models, in that it allows us to validate whether they hold, to parameterize them, or even consider new ways to approximate their assumptions.



(a) Public Cloud 2 (b) Entreprise cluster (c) Academic cluster

Figure 10: Network usage (for transmissions) as a function of the latency of individual RPCs

VII. Conclusions

In this paper we studied the latency of distributed protocols running on four common deployments. In the future, we intend to apply the lessons from this study to improve the latency of distributed protocols, namely by finding techniques to coordinate the activities of the protocols and the surrounding environment. Furthermore, our data can be used to automatically parameterize protocols that are based on recent proposals such as VFT [8] or XFT [9].

Acknowledgments. This work was supported by FCT with references UID/CEC/50021/2013 and UID/CEC/04516/2013. The research of R. Rodrigues was funded by the European Research Council (ERC-2012-StG-307732).

References

[1] R. Guerraoui, "Indulgent algorithms," in *Proc. of the 19th ACM PODC*, 2000, pp. 289–297.
[2] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *Proc. of the 7th USENIX OSDI*, 2006.
[3] P. Hunt, M. Konar, F. Junqueira, and B. Reed, "Zookeeper: wait-free coordination for internet-scale systems," in *Proc. of the USENIX ATC*, 2010.
[4] N. Lynch, *Distributed Algorithms*. Morgan Kaufmann, 1996.
[5] J. Leverich and C. Kozyrakis, "Reconciling high server utilization and sub-millisecond quality-of-service," in *Proc. of 9th ACM EuroSys*, 2014.
[6] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proc. of the ACM SIGCOMM*, 2011.
[7] K. Zarifis, R. Miao, M. Calder, E. Katz-Bassett, M. Yu, and J. Padhye, "Dibs: Just-in-time congestion mitigation for data centers," in *Proc. of 9th ACM EuroSys*, 2014.
[8] D. Porto, J. Leitao, C. Li, A. Clement, A. Kate, F. Junqueira, and R. Rodrigues, "Visigoth fault tolerance," in *Proc. of the 10th ACM EuroSys*, 2015.
[9] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolic, "XFT: practical fault tolerance beyond crashes," in *Proc. 12th USENIX OSDI*, 2016.
[10] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: Flexible, scalable schedulers for large compute clusters," in *Proc. of the 8th ACM Eurosys*, 2013.
[11] Y. Zhang, G. Prekas, G. M. Fumarola, M. Fontoura, I. Goiri, and R. Bianchini, "History-based harvesting of spare cycles and storage in large-scale datacenters," in *Proc. of the 12th USENIX OSDI*, 2016.
[12] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, and B.-G. Chun, "Making sense of performance in data analytics frameworks," in *Proc. of the 12th USENIX NSDI*, 2015.
[13] C. Smith and D. Henry, "High-performance linux cluster monitoring using java," in *Proc. of the 3rd LCIC*, 2002.
[14] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM TOCS*, vol. 20, no. 4, 2002.
[15] L. Lamport, "Paxos made simple," *ACM SIGACT News*, vol. 32, no. 4, 2001.
[16] A. Bessani, M. Santos, J. Felix, N. Neves, and M. Correia, "On the efficiency of durable state machine replication," in *Proc. USENIX ATC*, 2013.
[17] M. Maas, K. Asanović, T. Harris, and J. Kubiatowicz, "Taurus: A holistic language runtime system for coordinating distributed managed-language applications," in *Proc. of the 21st ACM ASPLOS*, 2016.
[18] R. Guerraoui and A. Schiper, ""γ-accurate" failure detectors," in *Proc. of the 10th WDAG*. Springer-Verlag, 1996.
[19] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *Proc. of the 1st Annual ACM SIGMM MMSys*, 2010.
[20] D. Bermbach and S. Tai, "Eventual consistency: How soon is eventual? an evaluation of amazon s3's consistency behavior," in *Proc. of 6th ACM MW4SOC*, 2011.
[21] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: Comparing public cloud providers," in *Proc. ACM IMC*, 2010.
[22] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: Observing, analyzing, and reducing variance," *Proc. VLDB Endow.*, vol. 3, no. 1-2, 2010.
[23] Z. Ou, H. Zhuang, J. K. Nurminen, A. Ylä-Jääski, and P. Hui, "Exploiting hardware heterogeneity within the same instance type of amazon ec2," in *Proc. 4th USENIX HotCloud*, 2012.
[24] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *Proc. of the 29th INFOCOM*. IEEE Press, 2010.
[25] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu, "Data consistency properties and the tradeoffs in commercial cloud storages: the consumer's perspective," in *Proc. of the 5th CIDR*, 2011.
[26] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, "Bobtail: avoiding long tails in the cloud," in *Proc. of the 10th USENIX NSDI*, 2013.
[27] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," in *Proc. of the 1st ACM WREN*, 2009.
[28] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. of the 10th ACM SIGCOMM IMC*, 2010.
[29] S. Kandula, S. Sengupta, A. Greenberg, and P. Patel, "The nature of data center traffic: Measurements and analysis," in *Proc. of 9th ACM IMC*, 2009.
[30] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, 1988.
[31] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J. ACM*, vol. 43, no. 2, 1996.
[32] F. Cristian and C. Fetzer, "The timed asynchronous distributed system model," *IEEE TPDS*, vol. 10, no. 6, 1999.
[33] P. Veríssimo and A. Casimiro, "The timely computing base model and architecture," *IEEE TC*, vol. 51, no. 8, 2002.