Kurma: Fast and Efficient Load Balancing for Geo-Distributed Storage Systems

Evaluation of Convergence and Scalability

Kirill L. Bogdanov *Waleed Reda*[1] *Gerald Q. Maguire Jr.* *Dejan Kostić* *Marco Canini*[2]
KTH Royal Institute of Technology [1]*Université catholique de Louvain* [2]*KAUST*

Abstract

This report provides an extended evaluation of Kurma, a practical implementation of a geo-distributed load balancer for backend storage systems. In this report we demonstrate the ability of distributed Kurma instances to accurately converge to the same solutions within 1% of the total datacenter's capacity and the ability of Kurma to scale up to 8 datacenters using a single CPU core at each datacenter.

1 Introduction

The increasing density of globally distributed datacenters reduces the network latency between neighboring datacenters and allows services deployed in neighboring locations to share workload when necessary, without violating strict Service Level Objectives (SLOs). This report presents an extended evaluation of Kurma, a practical implementation of a geo-distributed load balancer for backend storage systems. Kurma integrates network latency and a system's service time distributions to solve a decentralized rate-based performance model allowing global coordination among datacenters. For more details on Kurma see [6][1].

We integrated Kurma with Cassandra[4], a popular storage system. Using simulations and real world traces together with a geo-distributed deployment across Amazon EC2 [1] we demonstrate the ability of distributed Kurma instances to accurately converge to the same solutions (§2.2) within 1% of the total datacenter's capacity and the ability of Kurma to scale up to 8 datacenters using a single CPU core (§2.3).

2 Evalutaion

In this section, we aim to answer the following questions: (i) Do Kurma's geo-distributed instances converge to the same solution? (§2.2) (ii) How well can Kurma scale? (§2.3)

[1] Section 2.1, Table 1, and Fig.1 are the same as given in [6] as this describes the testbed and methodology that were used.

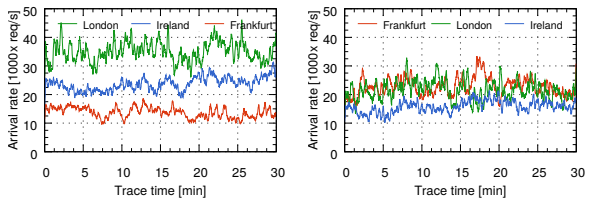
2.1 Evaluation Methodology

To evaluate Kurma we deployed Cassandra clusters across three geo-distributed datacenters of Amazon's EC2 located in Frankfurt, London, and Ireland. Each datacenter hosted up to 5 `r5.large` on-demand instances comprising the actual cluster and one `c4.4xlarge` instance running the YCSB workload generator [9]. The replication factor was set to 3 (each key is replicated 3 times in each datacenter). In all our experiments we assume eventual consistency — which is commonly used in practice [20, 14]. We use consistency level ONE for both reads and writes. Inline with the average value sizes found in production systems [5] we populate the database with 1 million keys that map to values of 150 bytes. The dataset was stored using Amazon's general purpose SSDs [3]. To minimize the impact of garbage collection on our measurements, we ran both Cassandra and YCSB instances on the Zing JVM [19].

Workload traces. We evaluate Kurma using real-world traces with temporal variations in workload (obtained from [2]). These traces represent a web-based workload and were recorded across multiple geo-distributed datacenters over an 88 day period. The traces show the rate of object requests per datacenter at one second resolution. For each second of a trace we fit a Poisson distribution to allow us to estimate the inter-arrival request rates at sub-second resolution. Table 1 shows the mapping between the original traces to the datacenter

Table 1: Mapping between the datacenters where a trace was initially recorded to the datacenter where it was replayed.

Src datacenter	Replayed in	Time shift (hrs)
Virginia	London	+0
Texas	Ireland	+1
California	Frankfurt	+4



(a) Trace-1: 147 M requests. 5-VMs per datacenter. (b) Trace-2: 105 M requests. 3-VMs per datacenter.

Figure 1: Two workload traces used in the evaluation.

where we have replayed them and the shift in time that we performed to correlate these traces with the timezones used for our experiment.

We modified YCSB to dispatch requests according to timestamps recorded in a trace file. For each experiment we verify that the workload generator is able to keep up with the required sending rate, thus acting as an open loop workload generator. Key popularity was distributed according to Zipf distribution.

For the evaluation on Amazon EC2, we selected two distinct intervals of 30 minutes (shown in Fig. 1). Both intervals were taken from a single day of the trace[2] and scaled by 450 to match the capacity of our hardware testbed while preserving workload variations. For brevity we refer to them as Trace-1 and Trace-2 respectively. Trace-1 demonstrates a major load imbalance, withe one of the datacenters more loaded than the rest. This provides an opportunity for load redirection. In contrast, in Trace-2, spare capacity is very limited and constantly shifts among datacenters, making load balancing challenging. To operate effectively in this trace a geo-distributed load balancer needs to recognize and act upon load balancing opportunities.

2.2 Do Kurma's geo-distributed instances converge to the same solution?

To evaluate the ability of distributed Kurma's instances to converge to the same solution, we measure the standard deviation of rate distribution decisions performed by Kurma's instances running in different datacenters. For Trace-1 experiments, we picked three load distribution rates that are actively changing. Fig. 2 shows how well distributed Kurma's instances agree on the rates that (i) Frankfurt should serve locally, (ii) Ireland should redirect to Frankfurt, and (iii) London should redirect to Frankfurt.

For most of the 30 minute trace, the standard deviation is under 500 req/s which is approximately 1% of the total capacity of one datacenter (with a cluster of five

[2]Day #49 [2] starting at 10:00 and 19:00 hours respectively based on Virginia's time zone.

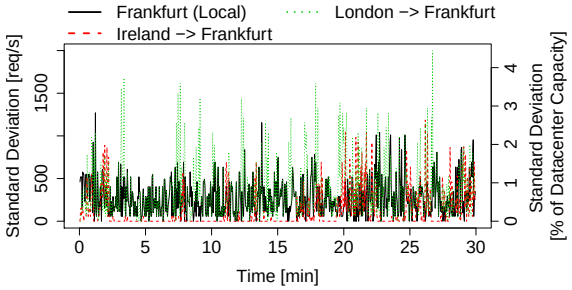


Figure 2: Standard deviation between the rates towards Frankfurt as seen from different from different Kurma instances.

Cassandra nodes as used for this experiment). From Fig. 2 we can see that the model outputs converge and on average this deviation is only 0.55% of the total capacity of a 5-VM cluster. This deviation is due to the stale load information at the distributed Kurma instances. Making load balancing decisions based on stale load information is known to be difficult [10] and can lead to suboptimal performance [17]. We obtain these results even though Kurma only exchanges load information at a 5 Hz frequency — showing that increasing the frequency of exchanges to greater than 5 Hz is largely unnecessary. We consider this to be a good result.

2.3 How Well Can Kurma Scale?

To assess Kurma's scalability, we empirically evaluate our model's time complexity. Fig. 3 demonstrates how the model computation times are affected by the size of the load balancing quantum and the number of datacenters. Using a load balancing quantum of 1% (default) Kurma solves the model for 5 datacenters, with a median completion time under 10 s. Using a load balancing quantum of 8% Kurma can scale up to 8

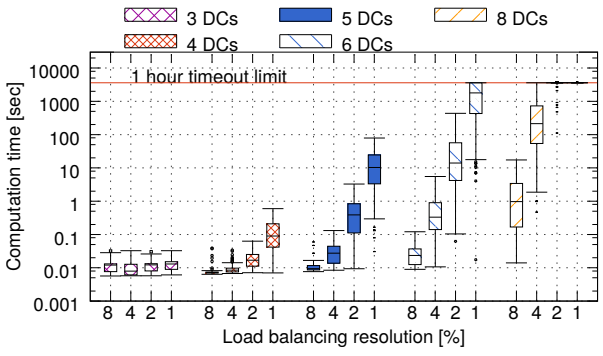


Figure 3: Model computation time for different load balancing quantum. Using only 1 CPU core, Kurma's model can support up to 8 datacenters.

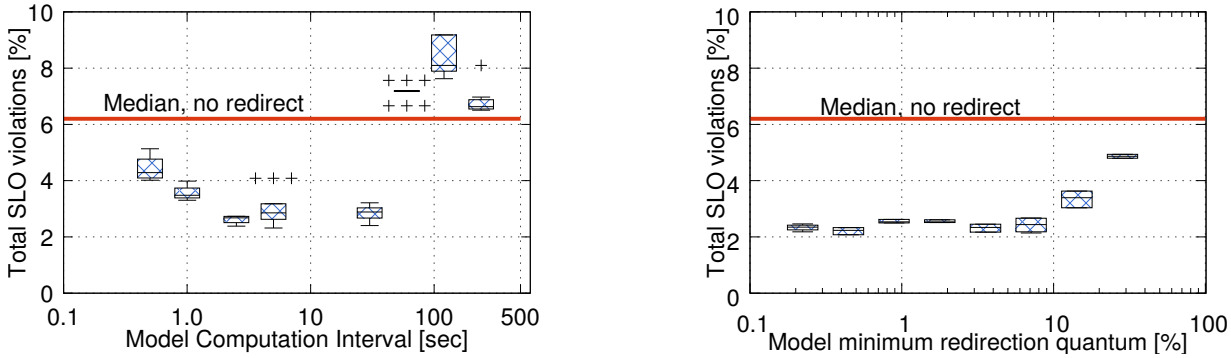


Figure 4: The effects of model recomputation intervals (left) and the size of the load balancing quantum (right) on Service Level Objective (SLO) violations.

datacenters with median completion time of 1 s.

Kurma computes its model sufficiently fast for today's scale of ten or fewer datacenters per provider. For providers that operate more datacenters, it is likely that only a fraction of them will be able to handle redirections while still satisfying SLOs (due to higher RTTs due to the large distance between them). As such, we posit that the number of datacenters that are viable targets for redirection (and therefore considered by our model) will remain in reach of our computational ability.

Next, using KVM emulation we evaluated the effects that changes in computation time and load balancing quantum have on Kurma's ability to reduce SLO violations. We utilized the same testbed settings with three datacenters, Wide area network (WAN) latency was emulated using the Linux `tc` command without jitter or packet loss and matched of the WAN latencies between the three EC2 datacenters.

In Fig. 4a we evaluate Kurma's sensitivity to different model recomputation intervals (over the range between 500 ms and 4 minutes; with the length of Kurma's averaging window set to double of the recomputation interval). In this experiment, our system tolerates computation intervals up to 30 s before performance starts to degrade. However, the desired frequency of model recomputation is a function of the load variability across geographic regions and can change over time.

Next, in Fig 4b we look at the effects of the load balancing quantum on the performance. We can increase the load balancing quantum up to 8% of the datacenter capacity without inflating SLO violations. Going beyond 8% makes Kurma more rigid, thus preventing it from agile load balancing, up to the point when no load balancing occurs. This indicates that by increasing the load balancing quantum we reduce the model's computation time, thus can increase Kurma's scalability *without* a major effect on performance.

3 Discussion

Can Kurma deal with different workload mixes in an online fashion? Service time distribution of a storage system can change non-negligibly depending on the workload characteristics, e.g., request sizes and read/write ratio. We assume, that in practice these characteristics will be known for the most common workloads, making it possible to incorporate them into the initial offline profiling. The process of offline profiling attempts to cover an expected range of workload characteristics. The configuration space is traversing with a variable step size, such that more samples are placed where the rate of SLO violations starts to grow rapidly (i.e., the knee of the SLO curve). We use linear interpolation to estimate the rate of SLO violations in between the closest configuration settings. Kurma can then react to changes in workload mixes by simply switching between pre-computed SLO curves as necessary.

Although, online adaptation to workloads that fall outside of the range of the offline profiling is currently not implemented, we do not consider this to be a limitation; as there is an extensive body of work that deals with run-time adaptation to changes in workloads [13, 15, 12, 7], if necessary, such techniques could be integrated with Kurma. Differences in workload sizes and mixes have been previously studied [18, 16] and are not main focus of this work.

Why implement Kurma as a distributed system? Centralized solutions introduce a single point of failure and can act as a potential target for security attacks. In contrast, implementing Kurma as a distributed system provides high availability and fits Cassandra's distributed architecture. Failure of any datacenter(s) will not prevent the remaining Kurma instances from performing

load balancing among the live datacenters[3]. While a centralized solution would eliminate minor divergences in among distributed instances of Kurma (previously shown in Fig. 2) it will also incur additional delay to collect and distribute information globally, especially in a setting where a service spans multiple geographic regions.

Can Kurma benefit from workload prediction techniques? In its current form, Kurma does not utilize any workload forecasting techniques. However, this is compensated for by the high frequency of model recomputations (in the order of a few seconds) which allows Kurma to rapidly correct allocation rates in response to dynamic changes in workload. Leveraging workload forecasting techniques [15, 11, 8], in between model recomputations, could further improve Kurma's ability to reduce SLO violations; however, this analysis is outside of the scope of this report.

4 Conclusion

In this technical report we evaluated Kurma's convergence and scalability. Using real world deployment across three geo-distributed datacenters of Amazon EC2, we showed that standard deviation in models' decisions across datacenters is under 1% of the datacenter capacity which we consider to be a good result. Next, we show that using a single CPU core at each datacenter, Kurma can compute its model sufficiently fast to be deployed across 8 datacenters, which is sufficient for many geo-distributed services. Furthermore, depending on the workload characteristics (i.e., load variability across geographic regions), the minimum load balancing quantum and the intervals between model recomputations can be increased; thus further facilitating Kurma's scalability.

References

[1] Amazon ec2. `http://aws.amazon.com/ec2/` Accessed 30-Nov-2015.

[2] The Internet Traffic Archive. `http://ita.ee.lbl.gov/html/contrib/WorldCup.html` Accessed 01-Feb-2018.

[3] AMAZON. Amazon elastic block store. `https://aws.amazon.com/ebs/details/` Accessed 01-Jun-2017.

[4] APACHE. Cassandra. `http://cassandra.apache.org/` Accessed 01-Feb-2018.

[5] ATIKOGLU, B., XU, Y., FRACHTENBERG, E., JIANG, S., AND PALECZNY, M. Workload analysis of a large-scale key-value store. In *ACM SIGMETRICS Performance Evaluation Review* (2012), vol. 40, ACM, pp. 53–64.

[6] BOGDANOV, K., REDA, W., MAGUIRE JR, G. Q., KOSTIĆ, D., AND CANINI, M. *Fast and Efficient Load Balancing for Geo-Distributed Storage Systems*. Submitted for review, KTH Royal Institute of Technology, 2018.

[3]Note, these properties do not necessary extend to the underlying storage system.

[7] BROSIG, F., HUBER, N., AND KOUNEV, S. Automated extraction of architecture-level performance models of distributed component-based systems. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering* (2011), IEEE Computer Society, pp. 183–192.

[8] CHANDRA, A., GONG, W., AND SHENOY, P. Dynamic resource allocation for shared data centers using online measurements. *ACM SIGMETRICS Performance Evaluation Review 31*, 1 (2003), 300–301.

[9] COOPER, B. F., SILBERSTEIN, A., TAM, E., RAMAKRISHNAN, R., AND SEARS, R. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing* (2010), ACM, pp. 143–154.

[10] DAHLIN, M. Interpreting stale load information. *IEEE Transactions on parallel and distributed systems 11*, 10 (2000), 1033–1047.

[11] GONG, Z., GU, X., AND WILKES, J. Press: Predictive elastic resource scaling for cloud systems. In *2010 International Conference on Network and Service Management* (2010), IEEE, pp. 9–16.

[12] HERBST, N. R., HUBER, N., KOUNEV, S., AND AMREHN, E. Self-adaptive workload classification and forecasting for proactive resource provisioning. *Concurrency and computation: practice and experience 26*, 12 (2014), 2053–2078.

[13] HERODOTOU, H., DONG, F., AND BABU, S. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *Proceedings of the 2nd ACM Symposium on Cloud Computing* (2011), ACM, p. 18.

[14] KALANTZIS, C. Eventual Consistency != Hopeful Consistency. talk at Cassandra Summit, 2013. `https://www.youtube.com/watch?v=A6qzx_HE3EU`.

[15] NGUYEN, H., SHEN, Z., GU, X., SUBBIAH, S., AND WILKES, J. Agile: Elastic distributed resource scaling for infrastructure-as-a-service. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)* (2013), pp. 69–82.

[16] NISHTALA, R., FUGAL, H., GRIMM, S., KWIATKOWSKI, M., LEE, H., LI, H. C., MCELROY, R., PALECZNY, M., PEEK, D., SAAB, P., ET AL. Scaling memcache at facebook. In *nsdi* (2013), vol. 13, pp. 385–398.

[17] RAGHAVAN, B., VISHWANATH, K., RAMABHADRAN, S., YOCUM, K., AND SNOEREN, A. C. Cloud control with distributed rate limiting. *ACM SIGCOMM Computer Communication Review 37*, 4 (2007), 337–348.

[18] SURESH, P. L., CANINI, M., SCHMID, S., AND FELDMANN, A. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *NSDI* (2015), pp. 513–527.

[19] TENE, G., IYENGAR, B., AND WOLF, M. C4: The continuously concurrent compacting collector. *ACM SIGPLAN Notices 46*, 11 (2011), 79–88.

[20] VENKATARAMANI, V., AMSDEN, Z., BRONSON, N., CABRERA III, G., CHAKKA, P., DIMOV, P., DING, H., FERRIS, J., GIARDULLO, A., HOON, J., ET AL. Tao: how facebook serves the social graph. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (2012), ACM, pp. 791–792.