Measuring and Characterizing the Performance of Interactive Multi-tier Cloud Applications

Mohammad Hajjat*, Shankaranarayanan PN, Ashiwan Sivakumar, Sanjay Rao
School of Electrical and Computer Engineering, Purdue University

***Abstract*—In this paper, we conduct a detailed study characterizing the performance of multi-tier web applications on commercial cloud platforms and evaluate the potential of techniques to improve the resilience of such applications to performance fluctuations in the cloud. In contrast to prior works that have studied the performance of *individual* cloud services or that of compute-intensive scientific applications (e.g., map-reduce based), our study focuses on multi-tier web applications. Our work is conducted in the context of four real-world web applications which we instrumented to collect the overall response time and the time spent in each application tier, for each transaction. Our results indicate that cloud applications undergo frequent periods of poor performance that typically (i) are short-lived lasting a few minutes; and (ii) may be attributed to a small subset of application components, though different subsets may be involved at different times. While geo-distributing applications can help mitigate performance variability, coarse-grained approaches that merely choose the best performing data-center (DC) provide only modest benefits. More significant benefits could accrue, however, if combination of cloud services located across multiple data-centers (DCs) are chosen to serve each request.**

***Index Terms*—**Cloud Computing, Interactive Multi-tier Applications, Monitoring Framework, Performance Variability, Request Redirection

I. Introduction

Cloud computing promises to reduce the cost of IT organizations by allowing them to purchase just as much compute and storage resources as needed, only when needed, and through lower capital and operational expense stemming from its economies of scale. Moving to the cloud is particularly attractive for applications given the potential for better scalability, resilience and disaster recovery.

While the advantages of cloud computing is triggering much interest among developers and IT managers, a key issue is meeting the stringent *Service-level Agreement (SLA)* requirements on response times for multi-tier web applications (e.g., customer-facing web applications, enterprise applications). Application latencies directly impact business revenue; e.g., Amazon found every $100ms$ of latency costs them 1% in sales [1]. Further, the SLAs typically require the $90th$ (and higher) percentile latencies to meet desired targets [2]. Given the shared multi-tenant nature of commercial cloud platforms, it is unclear whether such stringent SLA targets can be ensured.

In this paper, we present one of the first performance characterizations of multi-tier web applications on commercial cloud platforms. While researchers have studied the performance variability in cloud platforms, these works have either focused on *individual* cloud services (e.g., [3], [4]) or on high performance computing applications which primarily require minimizing completion time (e.g., [5], [6]). In contrast, our work focuses on user-facing and interactive web applications – an important class of applications that has received relatively little attention.

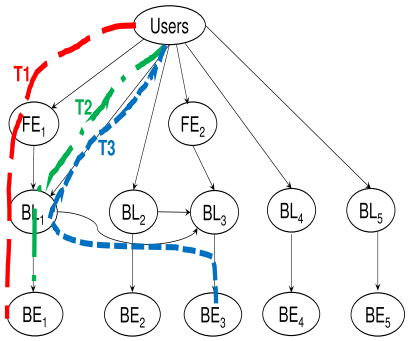


Fig. 1. Data flow of a multi-tiered Enterprise Resource Planning (ERP) application in a large university with thousands of users. *FE*, *BL* and *BE* denote the user-facing front-end, business-logic and back-end components, respectively.

Multi-tier applications typically have a large number of components with complex interaction patterns and multiple servers per component (Fig.1). We characterize the performance of such applications by measuring overall application response time and its constituent component processing times, and inter-component communication latencies. Breaking response time down into individual constituent delays is critical to better understand and diagnose application performance. We present a systematic methodology for instrumenting and monitoring multi-tiered applications at the granularity of individual requests. Our monitoring framework is designed such that it is online, easy to use with any cloud application and has minimal impact on the performance.

Using our framework, we characterize the performance of four different applications including two enterprise applications (*StockTrader* and *DayTrader*) that use conventional SQL databases, a social application (*Twissandra*) which uses a non-relational distributed database called *Cassandra* [7] and a data-intensive web application (*Thumbnail*). We evaluate the performance of these applications on two prominent cloud providers (Microsoft Azure and Amazon AWS) using realistic benchmarks over a two-day period.

Our results not only confirm that the high performance variability in the cloud is a key challenge in meeting application

*Now at Microsoft. Work done when at Purdue.

978-1-4673-6762-2/15/$31.00

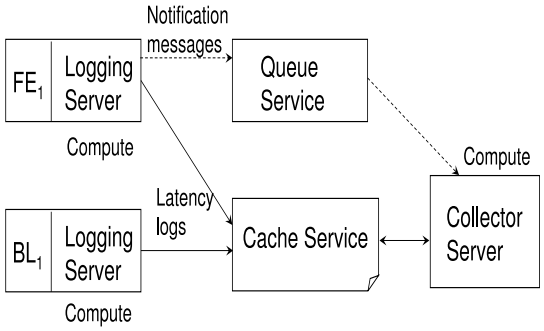


Fig. 2. Monitoring framework architecture for multi-tier applications. For simplicity we show two application components, FE_1 and BL_1.

SLAs, but also shed new light on the characteristics of bad performance periods. Almost 90% of the bad performance periods observed were short-term, lasting for less than 4 minutes and half of these episodes occurred within 4-9 minutes of each other. Interestingly, poor performance could typically be attributed to a small subset of application components, with different subsets being responsible at different times.

We next measure the performance of the applications *simultaneously* across two DCs and analyze the feasibility of mitigating performance variability by leveraging the geo-distributed nature of cloud applications. While the performance of component replicas across two DCs is uncorrelated as expected, surprisingly, we find that picking the best DC to serve each request in a coarse fashion provides only modest benefits. Our study indicates that much higher reductions in latency could accrue if a combination of component replicas (possibly located across multiple DCs) could be chosen to serve each request. E.g., for *Thumbnail*, choosing the best DC only reduces 95%ile (99%ile) latencies by 3.52% (6.82%). However, choosing the best combination of replicas reduces the 95%ile (99%ile) latencies by 13.37% (20.42%).

II. Measuring Application Performance

In measuring the performance of multi-tier applications, there are three main aspects to be considered. First, the measurements should be done at the granularity of individual application requests. While VM level performance counters (e.g., CPU usage) might be good indicators, it is non-trivial to translate such metrics into application performance. Second, it should be possible to isolate and attribute the performance of multi-tiered applications to its individual components to facilitate better diagnosis. Finally, the instrumentation framework should be easy to integrate with the application, impose minimal overhead and propagate the measurements as quickly as possible for analysis.

Black-box schemes for measuring performance (e.g., [8], [9]) are application agnostic, but employ statistical inference techniques which do not provide accurate measurements at the granularity of individual requests and components. *Annotation-based* schemes (e.g., Dapper [10], *X-Trace* [11]) tag every request with a global identifier for tracing its path in the application and thus provide accurate measurements at the required granularity. Dapper is tightly coupled with Google's RPC infrastructure. While *X-Trace* is more generally applicable, integrating it with multi-tiered applications is a manual and time consuming process. We leverage *X-Trace* but automate a large part of the integration effort using Aspect Oriented Programming (AOP) [12]– a technique for modularizing cross-cutting concerns in applications. The *X-Trace* code that has to be integrated with all function end-points in the application is separated into stand-alone modules – *aspects*. Using *aspects* to modularize code reduces the integration effort significantly. For instance, the *Source Lines Of Code (SLOC)* for instrumenting the *DayTrader* application is reduced from 1471 (without using aspects) to 366 (using aspects) which is almost a 75% reduction in SLOC.

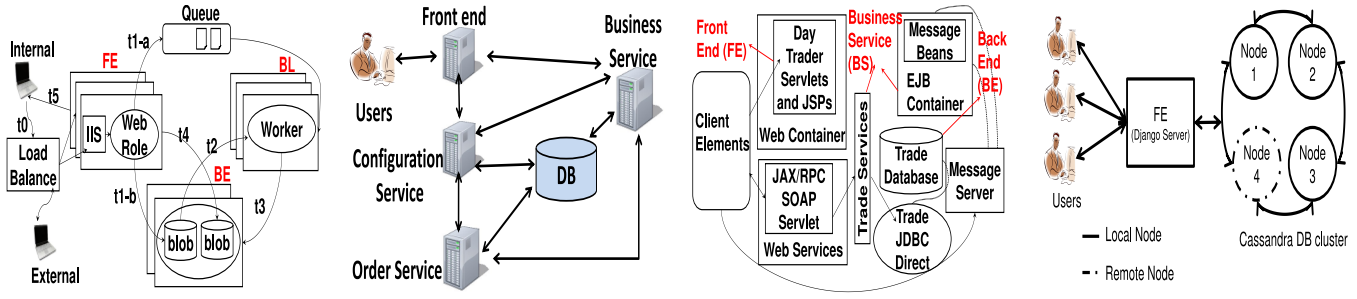
Fig. 2 shows the architecture of our monitoring framework. The instrumented application ships the measurement logs to a logging server (a process in the same compute instance) in order to minimize application overhead. The logging server in turn asynchronously moves the logs to a cloud storage. We use a cache service to store logs (Amazon ElastiCache, Azure Cache) since it is much faster than object stores (S3, Blobs), and data loss can be tolerated. The Collection Server (which performs the latency computations) is notified about the availability of measurement logs through a Queue Service (Amazon SQS, Azure Queues). Our framework has a small overhead on application latency; for e.g., the median response time of *DayTrader* shows an increase of only 10 milliseconds.

III. Applications and Test-bed Deployments

We choose four applications with different characteristics for our measurement study. These applications differ in their architecture, technology, components and interaction pattern between the components. *Thumbnail* [13] is a data-intensive web application for generating picture thumbnails. *StockTrader* [14] and *DayTrader* [15] are used for stock trading and represent widely used benchmark enterprise applications. *Twissandra* [16] is a social application which uses a distributed database system called Cassandra [7].

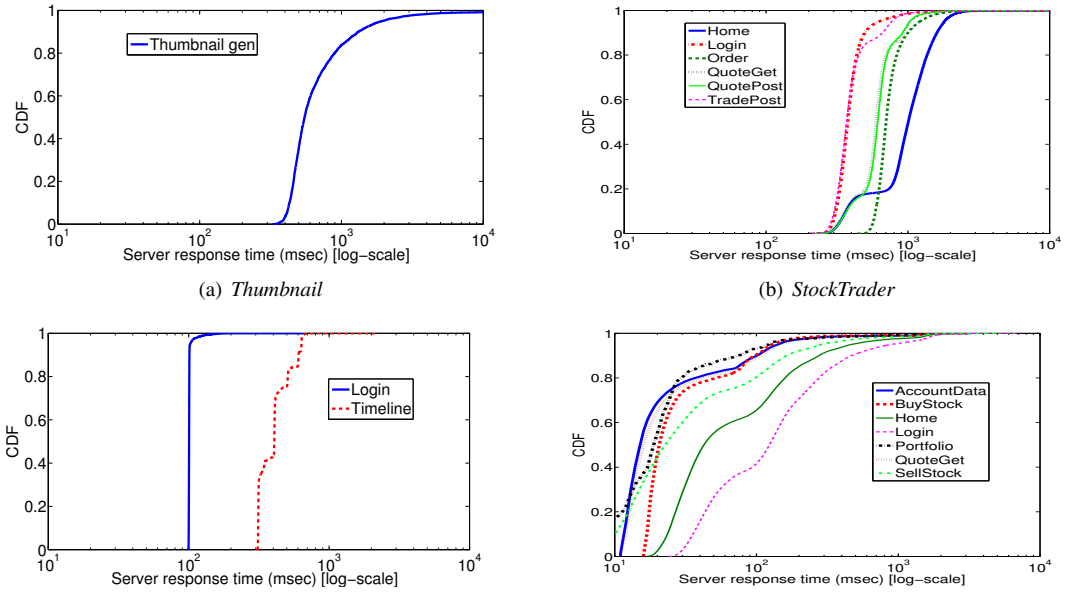
We deployed the applications on *Microsoft Azure* and *Amazon AWS* clouds. *Thumbnail* and *StockTrader* are .Net based applications and were naturally suited for deployment on *Microsoft Azure*, while *Twissandra* and *DayTrader* were deployed on *Amazon AWS*. We ran each application simultaneously in two separate DCs, both located in the U.S. The users were located in a campus network, also in the U.S. User requests were issued to both DCs simultaneously using the Grinder load testing framework [17], [18]. Our measurement study was conducted by load-testing each application continuously for a period of 48 hours.

The workload for *Thumbnail* consists of fixed size 1.4 MB images. The workloads for *StockTrader* and *DayTrader* were obtained from the associated DaCapo benchmarks [19]. The workload consists of several user sessions, each involving a series of transactions like login, view home page, view quote(s), buy/sell quotes, etc. For *Twissandra*, we made use of the Twitter Streaming API [20] to obtain a real data stream (Spritzer stream) to drive our experiments. We provision applications to handle the workloads by performing compre-



(a) *Thumbnail* (b) *StockTrader* (c) *DayTrader* (d) *Twissandra*

Fig. 3. Application architecture and data flow for the four applications. All applications have a front-end (FE), business logic server (BL/BS) and a back-end (BE). The BE is a Blob in (a), SQL databases in (b) and (c) and a Cassandra cluster in (d). In (b), the Order Service (OS) handles buy/sell operations, while the Config Service (CS) binds the various components together. In (d), the BS is coupled with the FE and hosted on the same server.



(a) *Thumbnail* (b) *StockTrader*

(c) *Twissandra* (d) *DayTrader*

Fig. 4. CDF of *server response time* for all applications, dissected by transaction types.

hensive stress tests with the same deployment configuration and workloads used in our experiments.

IV. Characterizing Application Performance

In this section, we measure the performance of multi-tier applications in the cloud and characterize periods of poor performance.

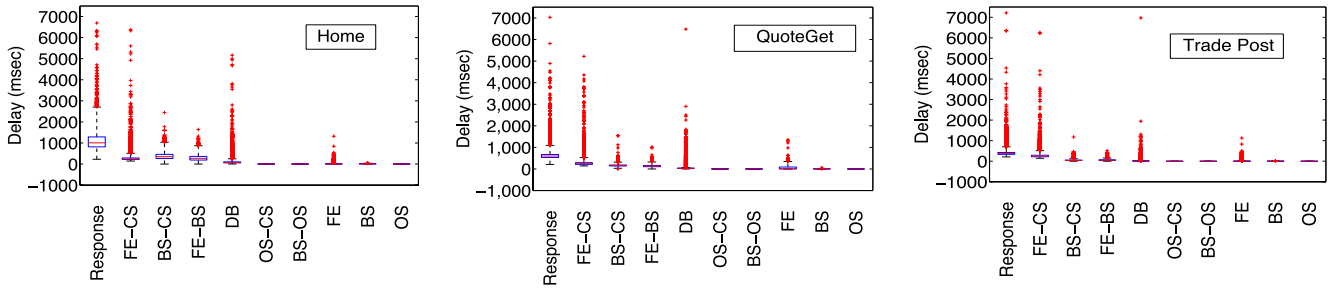
A. Decomposing application delays

Overall performance: Fig. 4 shows a CDF of the performance of all applications separated by different transaction types for the experiment described in §III. Since our focus is on cloud performance, we exclude Internet delays incurred between the user and the front-end servers of the applications. We refer to this delay as *server response time*. The figure shows that there is significant variation in *server response time* across all applications and transaction types. For example, the *coefficient of variation* ($\frac{standard\ deviation}{mean}$ ratio) for *Thumbnail* and *DayTrader* [*Login*] is about 6 and 2, while the ratio of 99.9%ile to median is about 232 and 44 respectively.

Dissecting *server response time*: Fig. 5 dissects *server response time* into its constituent ***elements*** (component processing times and inter-component communication latencies). It shows that while some *elements* show more variation than others (e.g., FE-CS, DB in *StockTrader*), in general there is variability in all elements.

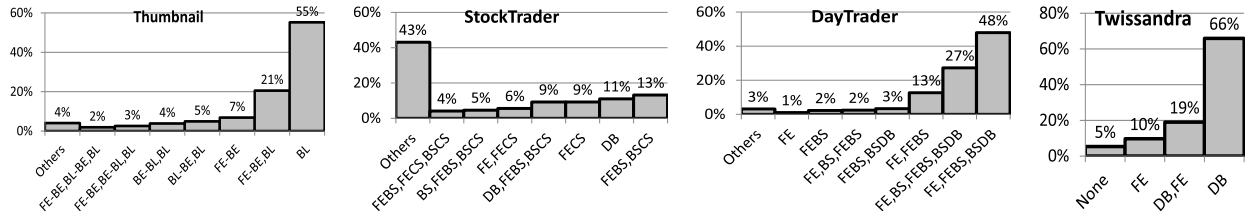
To identify the set of elements that are responsible for poor performance, we split the experiment data into 30 second time-windows. We consider *server response time* (constituent element) to be *high* during a window if its average during that window exceeds the 95%ile of *server response time* (constituent element) measured across the entire experiment. Fig. 6 shows the percentage of times each combination of elements was *high* when *server response time* was *high*. It can be observed that at any given time, only a subset of elements contribute to poor performance. Further, while in some applications, certain elements contribute to poor performance more frequently (e.g., DB in twissandra), in general, different subsets of elements affect the performance at different times across all applications.

Correlations across *elements*: Fig. 7 shows the correlation coefficients across all pairs of elements, for *StockTrader*. In



(a) Home transactions (b) Quote transactions (c) Post transactions

Fig. 5. Boxplots showing *server response time* and its constituent elements (processing and communication delays) for 3 transaction types for *StockTrader*. The bottom (top) of each box represent the 25%ile (75%ile), and the line in the middle represents the median. The *whiskers* extend to the highest datum within 3*IQR (inter-quartile range) of the upper quartile and points larger than this value are considered outliers.



(a) *Thumbnail* (b) *StockTrader* (c) *DayTrader* (d) *Twissandra*

Fig. 6. Percentage of combination of elements (i.e., processing and communication delays) that were *high* simultaneously when the *server response time* was *high*.

	FE	DB	BS	OS	FE-BS	FE-CS	BS-CS	BS-OS	OS-CS
FE	1	-0.08	-0.11	-0.04	-0.31	0.03	-0.32	-0.07	-0.04
DB		1	0.50	0.03	-0.01	-0.01	0.04	0.05	0.02
BS			1	0.14	0.08	-0.02	0.09	0.14	0.14
OS				1	-0.37	-0.03	-0.40	0.66	0.74
FE-BS					1	0.01	0.87	-0.31	-0.37
FE-CS						1	-0.01	-0.02	-0.03
BS-CS							1	-0.34	-0.41
BS-OS								1	0.71
OS-CS									1

(a) StockTrader

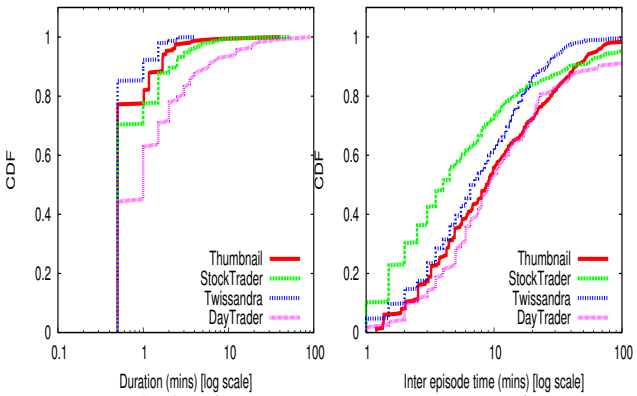
Fig. 7. Correlations across various *elements* of *StockTrader*. Values range between -1 (strongly anti-correlated) and +1 (strongly correlated).

general, there is little correlation in performance across elements. In cases where there is some correlation, we found that the degradation in the performance of a downstream element in the application graph affected the upstream elements (e.g., degradation of BS-CS affected FE-BS in *StockTrader*). Similar trends were also observed for other applications.

B. Characterizing episodes of bad performance

We characterize periods of bad performance seen by the applications and investigate their duration and frequency. We begin by identifying *Bad Performance Episodes (BPEs)*, where server response time is persistently *high* (as defined in §IV-A). However, since a period of *high* server response time may be interrupted by occasional non-*high* time-windows, a *BPE* is identified by marking groups of consecutive time-windows where atleast 70% of them have a *high* server response time.

Duration and frequency: Fig. 8(a) shows a CDF of durations of *BPE* for all applications. The figure shows that most of the performance problems that affect applications in the cloud are short-lived (90% of *BPE*'s last for less than 4 minutes). Fig. 8(b) shows a CDF of the inter-*BPE* time, measured as the time between the end of a *BPE* and the start of a new one. The figure shows that performance problems happen frequently within short intervals (half of the *BPE*'s occur within 4-9 minutes of each other, and that the 90^{th} percentile has a value of about 45 minutes or less). These results indicate that dynamic scaling based on addition of new server instances is inadequate since bringing up new server instances can take tens of minutes in commercial clouds.



(a) *BPE*'s durations (b) Inter-*BPE* time

Fig. 8. *BPE* duration and frequency.

Analyzing persistence: We use *auto-correlation* to analyze *persistence*– i.e., the tendency for *server response time* to remain in the same state from one time-window to the next. We use average *server response time* for every 30 second window to build the uni variate time-series used for auto-correlation.

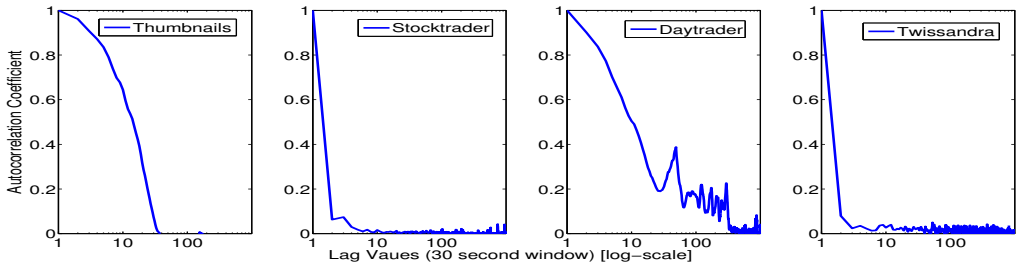


Fig. 9. ACF plots for *server response time* for all applications with max *time lags* of 500.

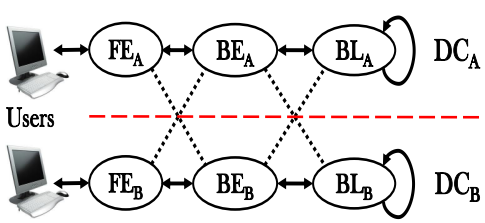


Fig. 10. *Thumbnail* deployment across DC_A (U.S. North Central) and DC_B (South Central). Nodes represent component replicas and lines represent communications between them. Solid (dotted) lines represent communications within (across) DC(s). There are 8 possible combination of replicas that can serve a request. A combination denotes the location of the FE, BE and BL component replicas, respectively (e.g., *ABA* indicates requests served by FE_A, BE_B and BL_A replicas, respectively).

We consider a 95% confidence bound (up to 5 lag values) for the analysis. In Fig. 9, we show the auto-correlation function (ACF) at varying *time lags* for all the applications. The figure shows that the average *server response time* has significant dependence with adjacent time-windows (and is thus predictable) for an interval of $1-10$ minutes ($2-20$ lag values) but becomes highly uncorrelated after that.

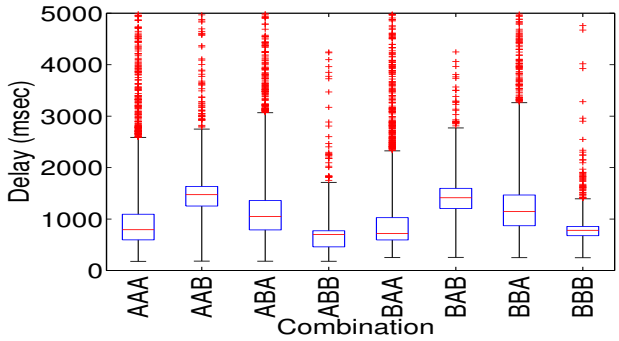
V. Exploiting Geo-distribution to Mitigate Performance Variability

In this section, we discuss new opportunities for tackling performance variability by leveraging the geo-distributed nature of cloud applications. Our insights are based on simultaneously measuring the performance of each application across two DCs. We observed that across all applications, the correlation coefficient for each *element* of *server response time* across the two DCs is low. E.g., for *StockTrader*, the correlation coefficients are 0.04 for *server response time* and 0.27, -0.01, -0.02 and 0.01 for the FE, BS, OS and DB processing delays respectively.

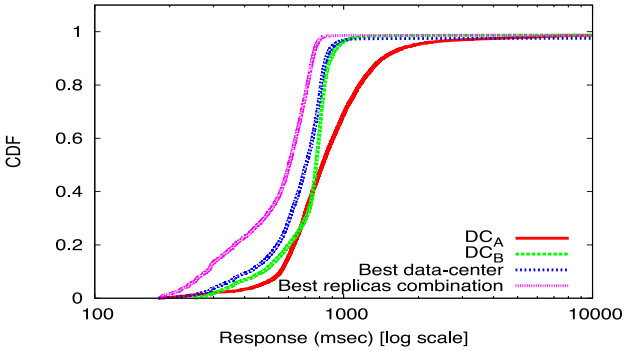
We now explore opportunities to exploit this lack of correlation to achieve better performance. Specifically, we consider two schemes:

1) ***Best DC*** selects the DC with better average *server response time* for each time-window. Each request is served entirely by a single DC. This is representative of coarse-grained schemes that load-balance application traffic across entire DCs as a whole (e.g., Akamai [21]).

2) ***Best combination*** selects the combination of replicas that achieves the best performance for each time-window. Note that replicas can be located in different DCs (Fig. 10). Choosing



(a) The *server response time* for all combination. Y-axis trimmed at 5 seconds.



(b) CDF of average *server response time*, calculated over 30 seconds time-windows.

Fig. 11. Exploiting geo-distribution to mitigate performance variability in *Thumbnail*.

replicas across different DCs is feasible since cloud applications are often built with *loosely-coupled* components [22].

To evaluate these schemes, we deploy the *Thumbnail* application in 2 DCs (Fig. 10). Requests along each combination of replicas were sent with equal probability. Fig. 11(a) shows a box-plot of *server response time* for all possible combination. Surprisingly, the delays of combination that cross DCs overlap significantly with combination restricted to one DC. E.g., the median *server response time* for AAA and BBB are 832 and 792 msec, respectively. Interestingly, both ABB and BAA have lower values of 724 and 735 msec, respectively.

To further analyze the potential of these schemes, in Fig. 11(b) we plot a CDF of average *server response time*, for each strategy. For reference, we also include the average *server response time* for the AAA and BBB combination. The *Best DC scheme* (best of AAA and BBB) reduces the 95%ile *server response time* by 3.52% over a scheme that always sticks to one DC. *Best combination*, on the other

hand, reduces it by 13.4%. Moreover, the improvements are even more pronounced for the 99%ile *server response time*. While *Best DC scheme* improves the performance by about 6.8%, *Best combination* achieves a 20.4% improvement. This shows that simply picking the best DC to serve requests does not produce the best performance. Instead, choosing the best combination of replicas across multiple DCs has the potential to achieve better performance.

VI. Related Work

CloudCmp [4] provides a systematic comparison of the performance and cost of storage, computing and networking services across different cloud providers. It aims to guide customers in selecting the best-performing provider for their applications. [3] characterizes the impact of virtualization on the networking performance of Amazon EC2. [23] develops micro-benchmarks of EC2 and introduces corresponding levels of background load in a laboratory-based cloud to measure the impact on latency of multimedia applications. Measurement studies like [6], [24] focus on the applicability of cloud computing for Many-Task Computing workloads used by the scientific computing community. Works like [5], [25] study the performance of map-reduce or High Performance Computing applications with task completion time as their primary metric. YCSB [26] provides a framework that facilitates performance comparisons of the new generation of cloud data serving systems like BigTable, Cassandra, Azure, SimpleDB, etc. Our work is distinguished from all these works in that our focus is on (i) measuring performance of applications in live cloud settings rather than *individual* cloud services; and (ii) studying multi-tier web applications rather than scientific computing applications. Further, we investigate ways to mitigate variability in the performance of geo-distributed cloud applications.

VII. Summary and Conclusions

In this paper, we have presented one of the first performance characterizations of interactive multi-tier applications deployed on commercial cloud platforms. Our results show that periods of poor application performance are (i) short-lived (90% lasting for less than 4 minutes); (ii) frequent (half of such periods occurring within 4-9 minutes of each other); and (iii) typically involve only a small subset of application components (though different subsets could be involved at different times). Our results also show that coarse-grained approaches that merely load-balance application traffic across entire DCs have limited benefits. In our experiments, while the 99%ile *server response time* could be reduced by 6.8% for *Thumbnail*, more significant benefits of up to 20% reduction could be achieved if different combination of cloud replicas located across multiple DCs are chosen to serve requests. In [27], we have presented an initial design of such a scheme that dynamically splits requests for each application component among its replicas in different DCs. The scheme addresses several practical issues including determining the right time-scale for adaptation, ensuring the system is responsive yet stable, estimating the performance of various possible combination while keeping overheads low, and ensuring the capacities of various component replicas are not exceeded.

VIII. Acknowledgments

This material is based upon work supported in part by the National Science Foundation (NSF) under Career Award No. 0953622 and NSF Award No. 1162333. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

References

[1] Latency - it costs you, http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it.
[2] D. Hastorun *et al.*, "Dynamo: amazons highly available key-value store," in *SOSP*, 2007.
[3] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of Amazon EC2 data center," in *IEEE INFOCOM 2010*.
[4] A. Li *et al.*, "CloudCmp: comparing public cloud providers," in *IMC 2010*.
[5] K. Jackson *et al.*, "Performance analysis of high performance computing applications on the amazon web services cloud," in *CloudCom*, 2010.
[6] A. Iosup *et al.*, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Parallel and Distributed Systems*, 2011.
[7] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS OSR*, vol. 44, no. 2, pp. 35–40, 2010.
[8] M. K. Aguilera *et al.*, "Performance Debugging for Distributed Systems of Black Boxes," in *Proc. SOSP*, 2003.
[9] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, "Magpie: Online modelling and performance-aware systems," in *HOTOS 2003*.
[10] B. Sigelman *et al.*, "Dapper, a large-scale distributed systems tracing infrastructure," *Google research*, 2010.
[11] R. Fonseca, G. Porter, R. Katz, S. Shenker, and I. Stoica, "X-trace: A pervasive network tracing framework," in *NSDI 2007*.
[12] Aspect Oriented Programming, http://msdn.microsoft.com/en-us/library/aa288717%28v=vs.71%29.aspx.
[13] Windows Azure Thumbnails Sample, http://code.msdn.microsoft.com/windowsazure/Windows-Azure-Thumbnails-c001c8d7.
[14] Stonehenge StockTrader Benchmark Application, https://cwiki.apache.org/confluence/display/STONEHENGE/Stonehenge+.NET+StockTrader+Installation+Guide.
[15] Apache DayTrader Benchmark Application, https://cwiki.apache.org/GMOxDOC20/daytrader.html.
[16] Twissandra, https://github.com/twissandra/twissandra.
[17] Grinder Load Testing, http://grinder.sourceforge.net/index.html.
[18] P. Nagpurkar *et al.*, "Workload characterization of selected jee-based web 2.0 applications," in *IISWC*. IEEE, 2008.
[19] S. M. Blackburn and et al., "The DaCapo benchmarks: Java benchmarking development and analysis," in *OOPSLA 2006*.
[20] Twitter Streaming APIs, https://dev.twitter.com/docs/streaming-apis.
[21] J. Dilley *et al.*, "Globally distributed content delivery," *Internet Computing, IEEE*, 2002.
[22] What's SOA got to do with cloud computing?, http://www-01.ibm.com/software/solutions/soa/newsletter/november11/whats_soa_got_to_with_cloud.html.
[23] S. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *MMSys 2010*.
[24] E. Walker, "Benchmarking amazon EC2 for high-performance scientific computing," *Usenix Login*, 2008.
[25] J. Schad *et al.*, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proceedings of the VLDB Endowment*, 2010.
[26] B. Cooper *et al.*, "Benchmarking cloud serving systems with YCSB," in *ACM symposium on Cloud computing*, 2010.
[27] M. Hajjat, S. P. N, D. Maltz, S. Rao, and K. Sripanidkulchai, "Dealer: Application-aware request splitting for interactive cloud applications," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012.