Management of SOA based context-aware applications hosted in a distributed cloud subject to percentile constraints

Keerthana Boloor*, Rada Chirkova†‡, Yannis Viniotis*‡
*Electrical and Computer Engineering,
†Computer Science,
North Carolina State University, Raleigh, NC, USA

***Abstract*—We consider geographically distributed datacenters forming a collectively managed cloud computing system. Multiple SaaS providers host their SOA-based, context-aware applications in the cloud. Typically, the context-aware applications serve multiple classes of customers (end users) classified on economic considerations, which determine the Quality of Service (QoS) received by each class. A QoS metric that has been explored in large distributed applications is the percentile of response times; this metric provides a form of guarantees on the shape of the response time distribution for the customer. This need for differentiated QoS for each customer class is incorporated into a Service Level Agreement (SLA) negotiated between the context-aware application provider and the cloud provider. Typical SLAs require the response time of a certain percentile of the input requests from particular classes of customers to be less than a specified value; if this value is exceeded, a penalty is charged to the cloud provider. In addition, the applications we consider are data-intensive with strict temporal order constraints that have to be enforced on requests within the same session of a customer. We propose Data-aware Session-grained Allocation with *g*i-FIFO Scheduling (DSA*g*S), a novel decentralized request management scheme deployed in each of the geographically distributed datacenters, to globally reduce the penalty charged to the cloud computing system. Our simulation evaluation shows that our dynamic scheme far outperforms commonly deployed management policies (typically employing static or random allocation with First In First Out, Weighted Round Robin or dynamic priority-based scheduling). We further optimize our solution for dynamic, data-intensive context-aware applications, by proposing a "context level" cache replacement policy. Our evaluation shows that, when used in conjunction with DSA*g*S, the replacement policy decreases the total penalty charged to the cloud.**

***Keywords*-Context-aware applications, Cloud computing system, SLA based management policies**

I. Introduction

Applications that are aware of the changes in their users' environment and react appropriately to them are said to be context-aware. The term "context" describes any information that characterizes the user's situation or any entity's situation the user could be interested in [1]. Context-aware applications thus provide smarter and more tailored responses to the user. These applications herald an era of ubiquitous computing where mobile customers can submit service requests from multiple devices, under different situations, and the application adapts to changes in the user's context.

Some examples of context-aware applications include location based services (for eg., applications that direct users to the closest vegetarian restaurant with less than 25 current patrons), high frequency trading applications where algorithms react to changes in stock values from different exchanges, remote health monitoring and rapid response applications which notify the earliest available medical personnel at the onset of a sudden critical condition in a patient.

As most context-aware applications mirror real-world behaviors, flexibility and extensibility are the most important requirements [2]. Over the past years, research efforts have focussed on the adoption of Service Oriented Architecture (SOA) principles to meet the requirements of context-aware applications. SOA based applications are composed of loosely coupled functional service end-points which can be instantiated on-demand and can be composed dynamically. Thus applications built on SOA principles are by design extensible and flexible making SOA a good fit for context-aware applications [3].

The most suited infrastructure for deployment of an enterprise level context-aware application is a cloud computing system which enables any hosted application to scale on-demand [4], offers "pay as you go" pricing and management services, reducing the total cost of ownership of the application.

Typically, customers of any large commercial application are geographically distributed. Enterprise application (SaaS) providers would prefer to cater to customers from geographically co-located datacenters due to improved response times and privacy issues. So cloud deployments tend to be global in nature with multiple geographically distributed datacenters forming the cloud and providing service to multiple SaaS providers each with application deployments in some or all of the datacenters.

A Service Level Agreement (SLA) is negotiated between a cloud provider and a SaaS provider which specifies terms and conditions of the service provided by the cloud for the application [4]. With the applications deployed across multiple datacenters, the SLA of each application is negotiated globally, across the geographically distributed datacenters.

Typically, SLAs for business applications specify (among other constraints) certain guarantees in terms of the fraction of requests serviced, as opposed to average-performance cri-

terion. Thus, many Service Level Agreements are designed to provide specific percentile-based performance goals. It has been shown in [5] that enforcing percentile performance criterion as a management objective, results in better conformance and improved response times in comparison to average performance criterion guarantees. Recent business trends in cloud computing systems have shown increasing adoption of fixed-step percentile SLAs, where a certain fraction of service requests for a hosted application is required to have a specific response time, if not a penalty is charged to the cloud [6]. As geographically distributed datacenters, each having a large number of servers, form a cloud computing system, hosting the applications, this percentile SLA has to be respected globally across all the servers among all the datacenters for the penalty to be minimized.

In our work, we consider a cloud computing system consisting of multiple geographically distributed datacenters, each with a large number of end-servers. The centers collectively host, multiple context-aware applications. The context-aware applications are each negotiated with percentile SLAs. We address the problem where the request management techniques employed by the cloud should aim at globally conforming to the percentile SLAs negotiated for each application, thus minimizing the penalty charged.

In summary, our contributions in this paper are:

- We propose DSA*g*S, a dynamic, distributed, adaptive, measurement based, data-aware, session-grained request management policy for geographically distributed datacenters hosting multiple SOA based context-aware applications with percentile SLAs, aiming to decrease the penalty charged to the cloud computing system.
- We perform extensive evaluations to demonstrate that our scheme provides improved allocations and schedules of requests at end-servers that aim to decrease the total penalty charged to the cloud globally, in comparison to the commonly deployed allocation and scheduling schemes.
- We propose a cache replacement policy for context-aware applications and evaluate the improvement of the percentiles achieved, owing to the replacement policy.

The paper is organized as follows. In Section II, we describe the problem we address and the system model under consideration. In Section III, we explain our request management technique. In Section IV, we list related work. In Section V, we evaluate our policy and compare it with alternatives.

II. Problem formulation

A. Context-aware application

In this paper we consider SOA based context-aware applications. We identify data dependency patterns and temporal order requirements common to a large number of context-aware applications. For example, consider an algorithmic trading or high frequency trading application that enables fast investments decisions. A typical request for such an application could require latest information about stocks of multiple companies from different stock exchanges and related information from risk management agencies [7]. Based on the most recent data, the application continually provides suggestions for investments.

The application described above needs to be context-aware, with responses tailored to reflect the current state of multiple aspects of the environment (the latest stock information from different exchanges, latest risk information from management agencies). This translates to the requirement of loading large amounts of data from multiple sources for servicing requests [8].

Also, in most real world context-aware applications, there is a strict timing order constraint and data dependency between requests from the same user/subject. In the high frequency FT application, for accurate execution of the trading algorithms, bids/offers placed/enquired by a particular user has to be executed in the same order as the requests were sent. In this work we refer to requests from the same customer/subject as belonging to the same **session**.

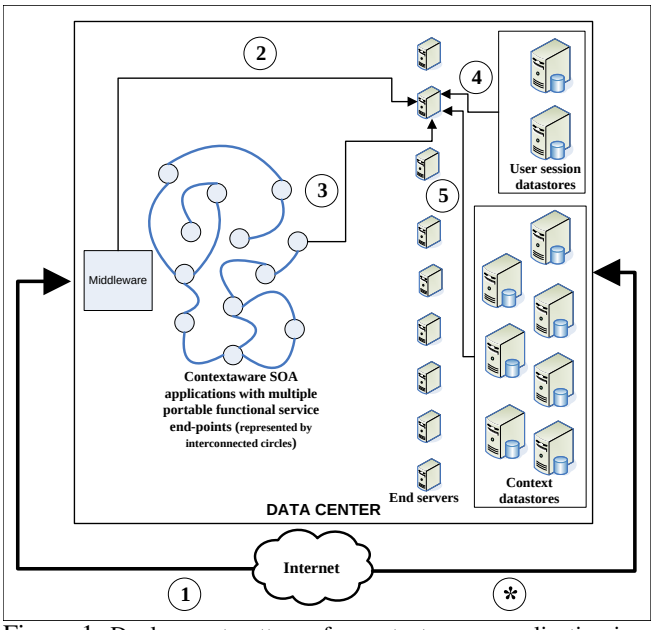

Figure 1: Deployment pattern of a context-aware application in a datacenter.

We now explain the typical operation that needs to occur when a request for the context-aware application considered, arrives (Steps 1 - 5 in Figure 1)

1) A request for the context-aware application is sent over the internet by the end-user.
2) The request is allocated to an end-server by the middleware.
3) The required service end-point for the functionality requested by the user is loaded on the chosen end-server.

4) The required session information is loaded on the chosen end-server from the session datastore. Subsequent requests within the same session are dependent on data from the output of processing previous requests from the same session (temporal order constraint).
5) The required context information from multiple context datastores is loaded on the chosen end-server. Multiple context datastores with data from different objects, are automatically updated with the latest information as shown in (*) in Figure 1.

A context-aware application may serve multiple classes of users typically classified using economic considerations. User classes that are designated more important than others need to be guaranteed a higher quality of service. So, the context-aware applications hosted by the cloud will need differentiated quality of service whose terms are negotiated in the Service Level Agreement.

A typical cloud computing system will provide hosting services to multiple software providers. In accordance, the geographically distributed cloud computing system we consider, hosts multiple context-aware applications, each of them serving multiple classes of end-users.

B. System architecture

The following are the key elements of the system:

- **Clients.** These are nodes that generate the service requests forwarded to the datacenters of the cloud.
- **Datacenters.** A datacenter is a cluster of a large number of networked computing resources. In the topology considered, multiple geographically distributed datacenters form the cloud computing system with each datacenter hosting the same set of context-aware applications. The requests for a context-aware application is served by the datacenter assigned to the end user's primary geographically location. This is a common assumption as sensitive data of a customer would need to be contained within a certain geographical boundary.

C. Step-wise percentile SLA

In this work, we consider a step-wise SLA where a penalty is charged to the cloud if a certain percentile of requests are not executed within a certain response time as shown in Figure 2. The SLA consists of multiple steps with each step associated with a percentile (also mentioned here as the threshold percentile) and a penalty value. As the fraction of requests meeting the response times increase beyond the threshold percentile of the first step, the penalty reduces to that of the second step and so on. The penalty is zero, if the fraction of requests meeting the response time increases beyond the threshold percentile of the last step.The **conformance levels**, i.e., the percentile of requests which have met the required response times is measured over a certain previously negotiated observation interval. The SLA is **global** in definition, i.e., all the datacenters in the cloud have to collectively respect the SLA.

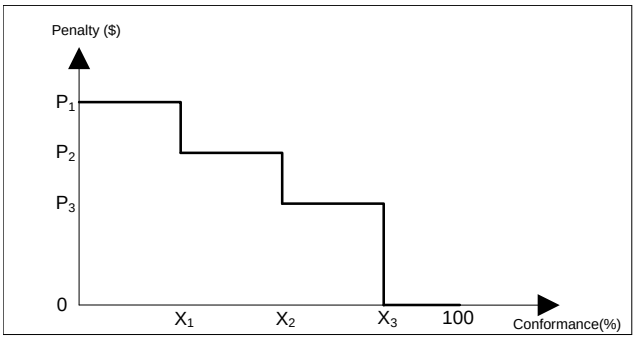

Figure 2: Multi-step percentile Service Level Agreement

The formal description of the SLA we consider is as follows: *Let n be the number of steps in the SLA* ($n = 3$ in Figure 2). *If the percentile of requests that have response time less than r seconds is less than or equal to $X_s\%$, then the cloud is charged a penalty of $\$P_s$, for $s = 1$ to n. If the percentile of requests that have the response time less than r seconds is greater than or equal to $X_n\%$, then no penalty is charged to the cloud. The percentile of requests are measured over a certain fixed observation interval (T seconds) and the penalty (if any) is charged for the observation interval.*

D. Objectives

A service request arriving at a datacenter has to be allocated to a particular end-server. Once allocated, the request has to be scheduled at the end-server. We need to propose a request management scheme that:

- performs differentiated allocation and scheduling based on the current conformance levels of the requests from different user classes of different applications in order to minimize penalty charged to the cloud,
- performs session-grained allocation to maintain strict order constraints and data dependency between successive requests in the same session,
- performs data-aware allocation. This is needed as, typically, the interval between consecutive updates to a context in e-business context-aware application is much larger than inter-arrival times of requests for the same context. This gives rise to the possibility of allocation of requests for the same context to the same end-server that has the required context data cached, greatly reducing response times as, for subsequent requests for the same context data, load times will be zero,
- balances load among the end-servers.

We make no assumptions about any prior knowledge about the number of requests arriving at the cloud computing systems for different applications or at different data centers.

E. Problem statement

We want to determine a request allocation and scheduling algorithm that provides the minimum in Equation 1 below:

$$\min \sum_{1 \leq k \leq K} \sum_{1 \leq j \leq C_k} penalty_{kj} \quad (1)$$

where $penalty_{kj}$ is the penalty charged for non-conformance of the requests from users of class j of application k; with the cloud hosting K context-aware applications each serving C_k classes of users.

III. Management policy description

The problem we consider belongs to the class of utility-maximizing, multi-class jobs, multiple resource allocation and scheduling problems. This class of problems has been proven to be NP-hard [9]. We propose heuristic based, greedy algorithms as part of our request management scheme with an aim to reduce the penalty charged to the cloud. Since the problem is NP-hard analytical proof that our policy achieves the least possible penalty cannot be derived. However, we compute divergence of our policy with the optimum by comparing the penalties obtained in our scheme with that obtained in an exhaustive search technique (feasible even in simulation for only small data sets) in an accompanying technical report [10].

A. Periodic exchange of updates

Since the penalty charged to the cloud is global in nature, each datacenter must be aware of the conformance levels of different classes achieved by all the other datacenters forming the cloud, in order to take suitable actions during allocation and scheduling requests. Recall that the penalty is charged for different classes, based on the conformance levels measured over an observation interval as given in Section II-C. We propose to divide the observation interval into multiple subintervals at each datacenter as shown in Figure 3. At the beginning of each subinterval, each datacenter exchanges the conformance levels of different classes with the peer datacenters that form the cloud. This exchange enables each datacenter to calculate the "current" **global** conformance levels of different classes. With "current" penalty, we mean the penalty charged to the cloud if the conformance level of the class is considered at the current instant. Our previous work [11] describes the periodic metric exchange in detail.

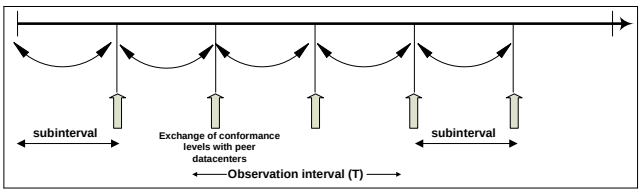

Figure 3: Observation interval divided into multiple subintervals in each datacenter

B. Request allocation

The allocation algorithm meets the objectives in Section II-D in the following fashion:

- Successive requests of the same session are always allocated to the same end-server. As mentioned in section II-A, each request from a customer belongs to a session. Typically, each request of context-aware application we consider will hold a unique id identifying the end-user [3] (Due to space limitations, we present a sample schema of XML based requests based on the OWL model for context-aware applications presented in [3] in an accompanying technical report [10]). This unique id is termed as session-id in this work. To allocate requests of the same session to the same end-server, we propose a hash-based lookup table containing tuples of the form ($session_{id}$,$server_{id}$). Each tuple in the lookup table represents a particular request(s) with session-id($session_{id}$) currently allocated on a particular end-server($server_{id}$). On each arriving request, the lookup table is queried to check if the $session_{id}$ of the arriving request is present, if yes, the request is allocated to the same end-server as in the tuple. The tuples are removed once requests with the same session-id are not received at the datacenter within a predetermined duration called the "session-validity period" which is a configurable parameter.
  There are a large number of efficient implementations of lookup-tables, especially targeting IP routing tables in literature [12]. The requirements of our lookup table are parallel to those of the IP routing tables. We consider a hash-based implementation of a dynamic lookup table with constant lookup $O(1)$.
  Allocating all requests of the same session to the same end-server satisfies the requirement of data dependency.
- If the requests's session-id is not found in the session lookup table (first request of the session), we perform differentiated allocation based on the "current" penalty charged by the class on the cloud as described in the following:
  1) A distributed hash table is queried to find the set of end-servers loaded with the some or all of the context data required by the incoming request (Implementation and complexity analysis is detailed in [10]).
  2) In all end-servers in the set obtained from the previous step, a "compatibility" test is performed to check if the incoming request can meet its response time in the end-server. If yes, the end-server is said to be **compatible** with the request. The compatibility test is based on the current penalty of the class of request and its deadline. A request of class with highest penalty is compatible in all end-servers, a request of class with lower

penalty with very high deadlines, might also be compatible in some of the end-servers that are lightly loaded or loaded with requests of classes with lower penalty. The compatibility test detailed in [10] derives heavily from the scheduling policy described in the next section.

3) Among all the end-servers found compatible, the request is allocated to the one with the most number of required contexts cached, satisfying the need for data-aware allocation. For facilitating data-aware allocation, our solution is influenced by [13], where authors propose a locality aware request distribution scheme for general purpose servers.
4) The estimation of completion time of a request at an end-server is greedy in nature. If a request of class with higher penalty is allocated to the same end-server, the same end-server may no longer be compatible for a previously allocated request.
5) If no machine is found compatible, which could be due to the request belonging to the class with low current penalty and/or has very small deadlines, the request is allocated to the least loaded end-server at the datacenter achieving load balancing.

- Requests belonging to classes with zero current penalty charged to the cloud are always allocated to the least loaded machine in the datacenter. This enables balancing of load at the various servers but not at the expense of reducing response times of requests of classes charging higher penalty to the cloud.

C. Request scheduling

We propose to model each end-server in a datacenter as having multiple queues, one for each user class. A request when allocated is inserted into the queue for its class at the chosen end-server. The aim of the scheduling policy at the end-server is to maximize the percentiles of the class of requests whose current conformance levels if unchanged would result in the highest penalty charged to the cloud. Since we adopt a greedy approach, we choose to schedule a request belonging to the class with the highest "current" penalty. Since strict temporal order constraints need to be maintained among requests of the same session, among the sessions queued of the chosen class, we choose the first request of a session which has waited the longest but whose deadline can still be met. If no such session exits, we choose a the first request of a session which has waited the longest. This method of choosing a class and then one session of that class, is based on the gi-FIFO scheduling policy which has been analytically proved in [14] to maximize percentiles for a single server serving multiple classes of jobs.

D. Session reallocation

A requirement of the allocation policy is to ensure the temporal constraints and data dependency between requests with the same session-id are maintained. To satisfy this requirement, we allocate requests with the same session-id to the same end-server at which the first request with the session-id was allocated without checking for compatibility with the end-server (see section III-B). This dependency is comparable to "allocation dependency" described in the context of HTTP sessions in [9]; the authors show a marked decrease in response times due to this dependency and propose a session-grained allocation approach. Similar to [9], in our simulations, we found that session "allocation dependency" resulted in reduced percentiles, details in section III-D. However, unlike in [9] (where authors propose a session-grained allocation scheme), in our problem there is no prior knowledge about number of requests in a session and request allocation decisions are taken immediately on their arrival.

If it were found that the end-server requests of a session were currently allocated to, will not be able to meet the deadlines of some or all requests in the session, we propose reallocation of all requests belonging to a session to another end-server. This reallocation occurs periodically, the interval between successive reallocations is configurable (Refer to section V). The periodic reallocation algorithm described and analyzed in [10] will migrate the session to the end-server in the datacenter capable of meeting the deadlines of the maximum number of requests of the session.

E. Context cache replacement policy

We consider data-intensive context-aware applications; the replacement policy for context caches affects the response times achieved. We propose a cache replacement policy for contexts loaded in each of the end-servers. For all the contexts cached on an end-server, we consider 3 parameters of interest, a) the normalized number of requests queued in the machine m referencing the context c (ref_{m_c}), b) the normalized highest penalty value among requests in the machine m referencing the context c ($penref_{m_c}$) and c) the normalized latest access time of the context c in machine m (acc_{m_c}). We also evaluate other parameters of interest and have detailed the results in [10]. Each of these parameters is associated with a "context cache coefficient" (ref,pen,$lacc$ ($\leq$ 1), respectively). We call the sum product of the three parameters with their respective context cache coefficients as the *cache replacement index* ($ref * ref_{m_c} + pen * penref_{m_c} + lacc * acc_{m_c}$). A coefficient of 1 of each of the three parameters assumes equal weight of each towards cache replacement index. A coefficient of 1 for the latest access time ($lacc$) and 0 for the others results in the LRU replacement policy. The context in the end-server with the lowest cache replacement index is replaced.

Note: Detailed assumptions in formulating the management policy, in [10].

IV. Related Work

To the best of our knowledge, we are the first to propose a solution for management of SOA based context-aware applications subject to percentile constraints. Our work brings together hitherto separate areas of interest, management of context-aware applications and enforcement of percentile constraints SLA for web-based applications. In this section, we consider objectives we aim to achieve in each area separately and compare them with previous research.

A. Qos in context-aware applications

In recent years, resource and/or request management in context-aware applications has garnered considerable interest. Huebscher and McCann [15] proposed a fault-tolerant adaptive middleware framework for context-aware applications which selects context-providers with an aim to maximize utility based on the accuracy of context-data from the providers. Lakshmanan et al. [16] address the problem of resource management in semantic event processing applications and propose a horizontal partition that is automatically created by analyzing the semantic dependencies among agents (service-endpoints) using a stratification principle. They implement a profiling-based technique for assigning agents to nodes in each stratum with the goal of maximizing throughput and distributing the load for increased scalability. Lorincz et al. [17] propose a new operating system for sensor nodes that enables resource aware programming while permitting high-level reusable resource management policies for context-aware applications. In comparison, we consider context-aware application built on SOA principles. We identify a need for dynamic data-dependency [8] and temporal order constraints requirement based allocation in a set of context-aware applications and propose an adaptive, data-aware, session-grained scheme for allocation of requests and aim to conform a performance metric(response times) to specified values. Our scheme can be implemented in a framework described by [17].

B. Percentile Service Level Agreements

Research on management policies for meeting SLAs citing performance percentile criterion has not not been as prolific as on policies for meeting average (mean) performance criterion. Gmach et al. [6] consider step-wise percentile SLAs and propose scheduling algorithms for a single database server unlike a distributed solution as proposed in this work. Xiong et al. [18] provide an analytical solution of resource optimization subject to percentile response time and price by modeling the system as an overtake free open tandem queuing network with feedback and provide closed form expressions of the probability distribution function of the response time. Cardellini et al. [5] present a brokering service for management of composite services under percentile-based SLAs; they propose a Qos model for composite services in which they provide an expression for the percentile of response time, assuming to know the α-quantile of the normalized response time in advance. To the best of our knowledge, these are the only 3 efforts for enforcing percentile SLAs in web-based applications. However, in contrast to the method described in the latter two schemes, we do not make any assumption about the distribution of input arrivals (or quantiles of the normalized response time) and service times, our allocation and scheduling schemes are measurement-based and adaptive in nature.

Adoption of learning techniques for utility maximizing adaptive resource management has been an active area of research in the recent years. Tesauro et al. [19] propose a reinforcement learning based management system for dynamic allocation of servers to web applications aiming to maximize the profit charged to the host datacenter. Reinforcement learning techniques seems most promising and investigation of applicability to this work is part of our future research.

V. Evaluation

We have developed a discrete event based simulator for evaluation of DSA*g*S. This section details simulation results to answer the following top-level questions:

1) How does DSA*g*S perform when compared to commonly deployed solutions?
2) Does session reallocation alleviate the "allocation dependency" problem?
3) Does the context replacement policy contribute to reduced penalties?

Due to space limitations, other questions of interest related to workload characteristics influencing DSA*g*S, effects of variations in periodic conformance exchange intervals to the penalties obtained etc., are examined in [10].

A. Comparison of DSAgS against commonly deployed schemes

A representative scenario involves a cloud computing system with (a) 3 context-aware applications hosted, serving 5 classes of users with uniformly distributed requests across datacenters, each negotiated with a step-wise percentile SLA as in Figure 2, (b) 5 geographically distributed datacenters, (c) 10 end-servers in each datacenter, (d) 100 distinct sessions and (e) 500 distinct contexts accessed during each observation interval, (f) each service request accesses data from 1-10 context-data sources with equal probability, (g) observation intervals of 1000 minutes, (h) context load times are uniformly distributed with a mean more than 3 times the average service rate (to model the data-intensive nature of the context-aware applications), (i) the input arrival process is Poisson, (j) the service processes are exponential. We compare DSA*g*S (without session reallocation) with commonly deployed schemes. The allocation schemes for comparison include:

- Static allocation - requests for certain context-data are always allocated to the same end-server. A static hash

function based on the context-data requested is used to map the requests to the end-server. Evaluations with variations in the static hash function is detailed in [10].

- Random allocation - requests are allocated to any end-server in the datacenter with equal probability.

The scheduling schemes at end-servers considered:

- FIFO (First In First Out)
- Dynamic priority with FIFO (Classes with highest "current" penalty always have the highest priority; priority is thus assigned dynamically. Once a high priority class is chosen, the request of the class that has waited the longest is scheduled (FIFO).)
- WRR (Weighted Round Robin with weights in proportion to the highest penalties of classes)

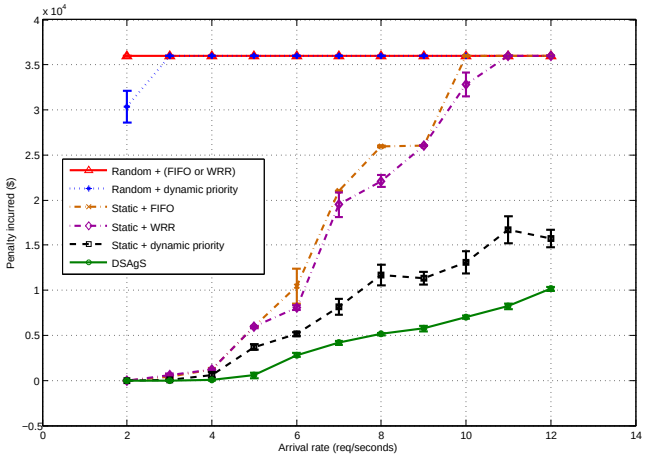

Figure 4: Comparison of DSA*g*S with commonly deployed solutions.

Typical results in Figure 4 show that DSA*g*S substantially outperforms the commonly deployed solutions (confidence intervals (ci) 95%). As expected, random allocation when employed with any of the scheduling policies, results in the lowest conformance and thus highest penalty (Random priority with FIFO or WRR resulted in highest penalties possible, indicated by the red solid line in Figure 4). As can be seen, a dynamic scheduling policy (priority) when used with a static allocation policy results in lower penalties (black dashed line in Figure 4) and so our dynamic allocation scheme with a dynamic scheduling policy designed to maximize percentiles improves the penalties further.

B. Evaluation of session reallocation

As mentioned in section III-D, we propose periodic reallocation of all requests in a session to a peer end-server in the datacenter, capable of meeting the deadline of the maximum number of requests in the session. Figure 5 compares the penalties obtained for DSA*g*S with and without session reallocation (ci 95%). In this experiment, a) the number of sessions is 10, b) the session reallocation interval at each end-server is 2 minutes, c) context load times are

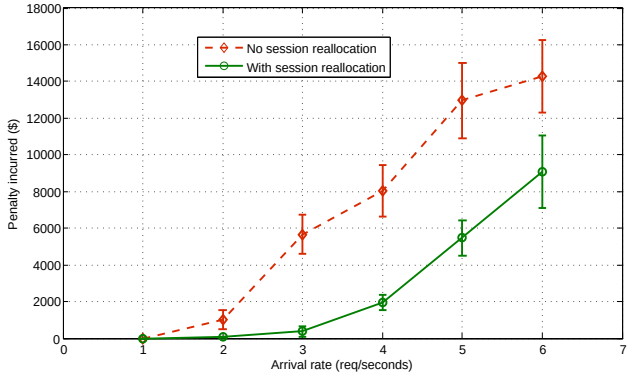

Figure 5: Comparison of DSA*g*S with and without session reallocation.

uniformly distributed with a mean more than 10 times the average service rate (The effect of performing periodic session reallocation is profound, when the context load times are much higher than the service rate (Refer to section III-D for details)).; the remaining attributes are as described in section V-A. As shown in Figure 5, the penalties obtained in DSA*g*S with session reallocation is lower when compared to DSA*g*S without session reallocation thus alleviating the effects of "allocation dependency".

Three parameters of interest in the evaluation of session reallocation are 1) the length of session reallocation period interval which influences the overhead caused, 2) the number of simultaneous sessions, and, 3) the number of requests in each session. Due to space limitations, the effect of variations of these parameters on the penalties obtained when employing session reallocation is detailed in [10].

C. Evaluation of context replacement policy

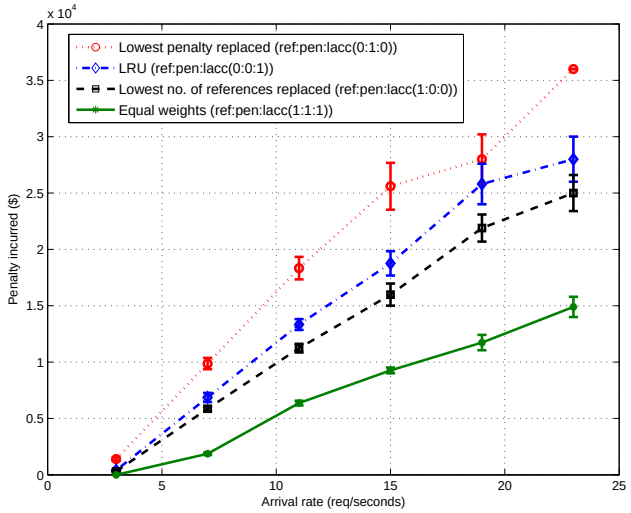

Figure 6: Penalties charged for different context cache coeffecient combinations.

We propose a context-cache replacement policy described in section III-E. We varied the context cache coefficients

(ref,pen,$lacc$) and observed the effect on the penalties charged. In this experiment, we maintained context load times to be uniformly distributed with a mean more than 10 times the average service rate in order to determine the effect of caching on the penalties. The remaining system parameters are same as in section V-A. Figure 6 shows that an equal value for all context cache coefficients results in lower penalties (ci 95%). In [10], we show that the context cache coefficients are sensitive to variations in the workload and input arrival characteristics and coeffecient values need to be chosen appropriately for significant reduction in penalties charged to the cloud.

VI. Conclusion and Future Work

In this paper we studied the problem of management of SOA based, data-intensive context-aware applications hosted in a distributed cloud where the system operates under a global, percentile response time SLA. The SLA calls for economic penalties if percentile targets are not met. We proposed Data-aware Session-grained Allocation with ***g***i-FIFO Scheduling (DSA***g***S), a novel decentralized request management scheme. Our simulation evaluation shows that our dynamic scheme far outperforms commonly deployed management policies in achieving lower penalties. We also proposed and evaluated a "context level" cache replacement policy that contributes to reduced penalties charged to the cloud provider hosting the context-aware applications.

Our future work includes (a) expanding the scope of the problem to include estimation errors of processing and data access, (b) evaluation of our solution when implemented in real world distributed data-intensive context-aware applications, (c) investigations into applicability of learning techniques to obtain a generic solution to the percentile conformance problem.

References

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, ser. HUC '99. Springer-Verlag, 1999, pp. 304–307.

[2] M. Ebling, G. Hunt, and H. Lei, "Issues for context services for pervasive computing," in *Proceedings of Middleware01, Advanced Workshop on Middleware for Mobile Computing*, November 2001.

[3] T. Gu, H. K. Pung, and D. Q. Zhang, "A service-oriented middleware for building context-aware services," *J. Netw. Comput. Appl.*, vol. 28, pp. 1–18, 2005.

[4] L. Zhang and D. Ardagna, "Sla based profit optimization in autonomic computing systems," in *ICSOC '04*. ACM, 2004.

[5] V. Cardellini, E. Casalicchio, V. Grassi, and F. L. Presti, "Adaptive management of composite services under percentile-based service level agreements," in *ICSOC*, 2010.

[6] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, and A. Kemper, "Adaptive quality of service management for enterprise services," *ACM Trans. Web*, vol. 2, no. 1, pp. 8:1–8:46, 2008.

[7] M. V. Sewell and W. Yan, "Ultra high frequency financial data," in *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, ser. GECCO '08. ACM, 2008, pp. 1847–1850.

[8] G. Soundararajan, M. Mihailescu, and C. Amza, "Context-aware prefetching at the storage server," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*. USENIX Association, 2008, pp. 377–390.

[9] X. Tang, S. T. Chanson, H. Chi, and C. Lin, "Session-affinity aware request allocation for web clusters," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, ser. ICDCS '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 142 – 149.

[10] K. Boloor, R. Chirkova, and Y. Viniotis, "Management of soa based context aware applications hosted in a distributed cloud subject to percentile constraints," North Carolina State University-Department of Computer Science, Tech. Rep. Dummy number, February 2011.

[11] K. Boloor, R. Chirkova, Y. Viniotis, and T. Salo, "Dynamic request allocation and scheduling for context aware applications subject to a percentile response time sla in a distributed cloud," *IEEE Cloudcom*, 2010.

[12] C. J. Martinez, D. K. Pandya, and W.-M. Lin, "On designing fast nonuniformly distributed ip address lookup hashing algorithms," *IEEE/ACM Trans. Netw.*, vol. 17, no. 6, pp. 1916–1925, 2009.

[13] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-aware request distribution in cluster-based network servers," *SIGOPS Oper. Syst. Rev.*, vol. 32, pp. 205–216, October 1998.

[14] N. Agarwal and I. Viniotis, "Performance space of a gi/g/1 queueing system under a percentile goal criterion," in *MASCOTS '95*. IEEE Computer Society, 1995, pp. 474–484.

[15] C. Huebscher and A. McCann, "An adaptive middleware framework for context-aware applications," *Personal Ubiquitous Comput.*, vol. 10, no. 1, pp. 12–20, 2005.

[16] G. T. Lakshmanan, Y. G. Rabinovich, and O. Etzion, "A stratified approach for supporting high throughput event processing applications," in *DEBS '09*. ACM, 2009, pp. 5:1–5:12.

[17] K. Lorincz, B.-r. Chen, J. Waterman, G. Werner-Allen, and M. Welsh, "Resource aware programming in the pixie os," in *SenSys '08*. ACM, 2008, pp. 211–224.

[18] K. Xiong and H. Perros, "Sla-based service composition in enterprise computing," in *16th International Workshop on Quality of Service, IWQoS*. Washington,DC,USA: IEEE Computer Society, 2008, pp. 30 –39.

[19] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A hybrid reinforcement learning approach to autonomic resource allocation," in *In Proc. of ICAC-06*, 2006, pp. 65–73.