Feedback-based Scheduling for Back-end Databases in Shared Dynamic Content Server Clusters

Gokul Soundararajan, Kaloian Manassiev, Jin Chen, Ashvin Goel, Cristiana Amza
University of Toronto, Canada
{gokul,kaloianm,jinchen,ashvin,amza}@eecg.toronto.edu

Abstract

This paper introduces a self-configuring architecture for scaling the database tier of dynamic content web servers. We use a unified approach to load and fault management based on dynamic data replication and feedback-based scheduling. While replication provides scaling and high availability, feedback scheduling dynamically allocates tasks to commodity databases across workloads in response to peak loads or failure conditions thus providing quality of service. By augmenting the feedback loop with state awareness, we avoid oscillations in resource allocation.

We investigate our transparent provisioning mechanisms in the database tier using the TPC-W e-commerce and the on-line auction Rubis benchmarks. We demonstrate that our techniques provide quality of service under load bursts and failure scenarios.

1   Key Contributions

This paper introduces a novel scheduling technique for on-demand resource allocation across multiple dynamic-content workloads that use a cluster-based database back-end. Dynamic content servers commonly use a three-tier architecture that consists of a front-end web server tier, an application server tier that implements the business logic, and a back-end database tier that stores the dynamic content of the site. Our dynamic-content cluster architecture consists of a set of schedulers, one per workload, that distribute incoming requests to a cluster of database replicas and deliver the responses back to the application server. The application server interacts directly only with the scheduler in charge of the corresponding workload run by the application server. In addition, a controller arbitrates resource allocations between the different workloads.

We define quality of service as maintaining the average query latency for a particular workload under a pre-defined Service Level Agreement (SLA). Our dynamic database provisioning algorithm, called feedback-based scheduling (FBS) [2] triggers adaptations in response to impending SLA violations. Furthermore, our algorithm removes resources from a workload's allocation when in underload. Due to the state-full nature of databases, the allocation of a new database to a workload requires the transfer of data to bring that replica up to date. We use an adaptation scheme called *warm migration* where: i) All databases in the workload's main partition are kept up-to-date and ii) We maintain a set of additional replicas within a staleness bound. These replicas constitute an overflow pool used for rapid adaptation to temporary load spikes. Write-type queries are batched and sent periodically to update overlap replicas whenever they violate the staleness bound.

To meet the SLA, FBS uses two key components: (1) per-workload performance monitoring, and (2) system-state awareness through a state machine approach. Average latency sampling is used to trigger adaptations in response to impending SLA violations. At the same time, the controller uses a state machine approach to track the system state during and in-between adaptations in order to trigger any subsequent adaptations only after the changes of all previous adaptations have become visible. Latency sampling is thus suppressed while an adaptation is in progress (e.g., during data migration to bring a new replica up to date) because effects on latency cannot be reliably observed. This closes the feedback loop and avoids unnecessary over-reaction and oscillation in the system.

Our experimental evaluation uses the TPC-W industry-standard e-commerce benchmark [3] modeling an on-line book-store and Rubis, an on-line auction site modeled after e-Bay [1]. Our results show that our feedback-based approach can handle rapid variations in an application's resource requirements while main-
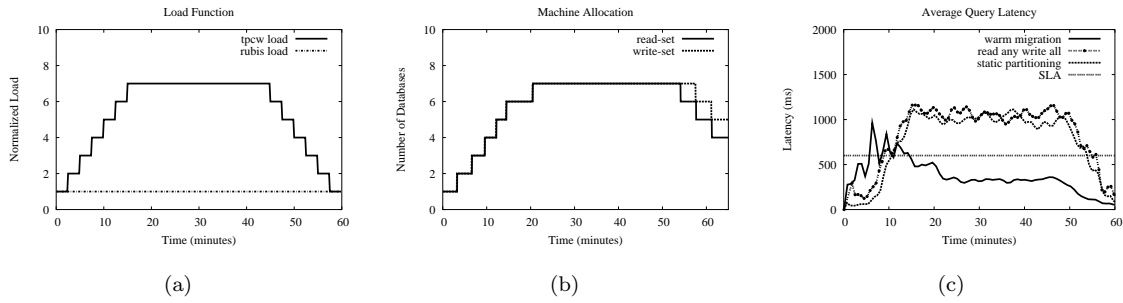
Figure 1. Dynamic versus Static Partitioning and Static Read-one-Write-all Resource Allocation
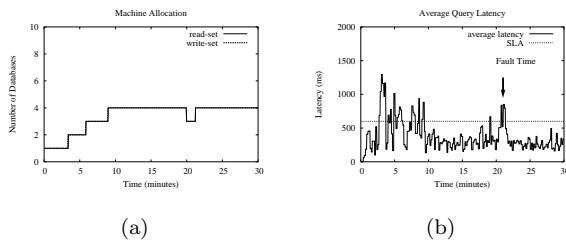

Figure 2. Adaptation to Load and Failures

taining quality of service across applications. In addition, by monitoring system state, we avoid oscillations in resource allocation. Finally, we show that the same approach can be used to handle failure scenarios.

2 Evaluation

We use a Linux cluster of AMD Athlons connected by Fast Ethernet. We run TPC-W and Rubis using a SLA of 600 ms for the database query latency which is conservatively chosen in order to keep the client latency below 1 second for each workload. Our state-aware protocol allows us to avoid careful tuning of the sampling interval and any other system parameters. We can thus use a sampling interval of 10 seconds with its implied high potential reactivity to load changes.

2.1 Adaptation to Increasing Load

Figure 1(a) shows our adaptations to load in comparison with two static algorithms: a static partitioning algorithm and a read-any-write-all algorithm. Figure 1(a) shows TPC-W under a seven-fold load variation while Rubis is kept constant at level 1. For each workload, the client load representation is normalized to the number of clients needed to saturate

one database. The overall result shown in Figure 1(c) is that our system performs well as a result of its flexible machine allocation (Figure 1(b)) with very infrequent and brief SLA violations. Static partitioning violates the SLA due to insufficient resources and the poor performance of read-any-write-all algorithm is explained by the interference of the read sets of the two workloads in the buffer cache of the database. We also see that our machine allocation closely follows the load pattern with on-demand machine allocations and releasing databases when in underload and no oscillations.

2.2 Adaptation to Failures

We use the same load function from the previous section (Figure 1(a)) but induce a fault at the 20 minute mark of the experiment. As the load for TPC-W increases, the number of machines allocated to handle the load also increases. After 20 minutes of running time, we induce a fault in the one of the databases used by TPC-W. Due to the database fault, the latency of TPC-W steadily increases from 300 ms until it violates the SLA as shown in Figure 2(b). At this point, the controller adds a new database and the latency drops to pre-fault levels.

References

[1] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. El-nikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Specification and implementation of dynamic web site benchmarks. In *5th IEEE Workshop on Workload Characterization*, November 2002.

[2] G. Soundararajan and C. Amza. Autonomic provisioning of backend databases in dynamic content web servers. Technical Report TR05-06, Electrical and Computer Engineering, University of Toronto, January 2005. http://www.eecg.utoronto.ca/ gokul/chameleon.html.

[3] Transaction Processing Council. http://www.tpc.org/.