Elastic cloud storage control for non-uniform workloads

Nicholas Trevor Rutherford

Master of Science Thesis
Stockholm, Sweden 2012

TRITA-ICT-EX-2012:163

Elastic cloud storage control for non-uniform workloads

EMDC Masters thesis of Nicholas Trevor Rutherford, KTH[1] & UPC.

July 2012

[1]School of Information and Communication Technology. Supervisors: Ahmad Al-Shishtawy and Associate Professor Vladimir Vlassov. Examiner and mentor: Associate Professor Johan Montelius

Abstract

Storage systems are a critical component of the 3-tier web applications increasingly deployed to cloud computing platforms. Elasticity is the cloud's most marketable attribute, and in order to achieve this for storage systems automatic control is required to achieve self-management.

This work investigates partition-demand aware control, demonstrating experimentally that this information can be turned to control able to consider the structure of the workload it is observing, rather than assuming it evenly distributed across the keyspace. This enables fine-grained load-balancing, leading to a reduction in cloud infrastructure rented by a self-managing storage system.

Experimental results with the (Dynamo based) Voldemort key-value store demonstrate the functionality of this control mechanism for pathological examples, and lead the way for future work or integration with more sophisticated storage control systems.

Contents

Chapter 1

Introduction

Storage systems are a critical component of the 3-tier web applications increasingly deployed to cloud computing platforms. Elasticity is the cloud's most marketable attribute, and in order to achieve this for storage systems automatic control is required to achieve self-management.

This work investigates partition-demand aware control, demonstrating experimentally that this information can be turned to control able to consider the structure of the workload it is observing, rather than assuming it evenly distributed across the keyspace. This dichotomy is illustrated by Figure 1.1. This enables fine-grained load-balancing, leading to a reduction in cloud infrastructure rented by a self-managing storage system.

Motivation for cloud storage research

Elasticity is a key selling point for cloud computing. Paying only for what you need, and being able to immediately shrink or grow your rented infrastructure according to changes in demand, is very attractive to companies whose applications have unstable loads, such as new applications which may "go viral" and see huge increases in popularity during a short period of time.

While IaaS platforms provide utility-model infrastructure rental, this enables but does not achieve the system elasticity needed to take advantage of it. Dynamic resource management is required to adapt the consumed resources according to demand, whether provided by a human overseer or the system itself. Self-managing systems are able to monitor themselves and respond to change, achieving goals such as fault-tolerance, consistent performance, or cost optimisation, all without human intervention. Such systems may respond faster than human operators, cost less to employ in achieving always-on monitoring, and promise to manage complex systems beyond the understanding of human operators.

Three-tier web applications are the focus of much industrial and academic attention, comprising the vanguard of cloud adopters and being central to many new business ventures. These applications typically require state persis-

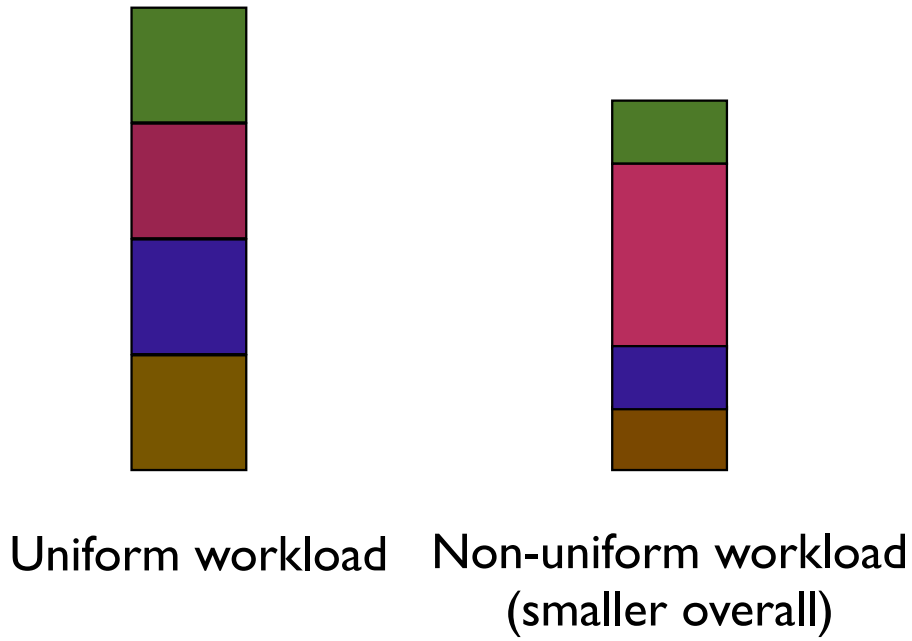

Figure 1.1: Uniform and non-uniform partition-load histogram stacks, where each block is a data partition's access rate, and the stack's total height is the system workload.

tence, to retain user data or other information to present to site visitors. These applications must scale with demand, and in doing so each of the tiers must also scale. Such applications provide a number of features making their control useful and interesting. They are are constrained by infrastructure rental costs, data transfer costs, and may be governed by contractual constraints in the form of SLA, dictating a required quality of service such as the average or percentile-based latency.

The storage tier of an application is particularly tricky, as aside from the well-known issues of consistency, partitioning, fault-tolerance and concurrency control, which we do not address herein, the state-transfer required when changing the storage cluster size complicates its control. This data transfer has two profound effects on the storage service: it worsens its performance, and it delays the onset of benefits from scaling.

Despite these complexities, human managers are reluctant to surrender control of their systems to automatic controllers without confidence that they are safe and effective – reliably reducing their operating costs without violating their service level agreements. However, existing systems have been received with scepticism [1]. In order to exploit the benefit of elastic scaling offered by cloud computing, we need to investigate mechanisms for building systems which autonomously manage their own resource consumption.

Contributions

Past work (section 3) has considered methods for determining the resources which should be made available to a storage system, but entrusts the system with the task of efficiently utilising those resources.

In this work we present a control mechanism for solving this second problem, of how to allocate data to storage nodes, so that performance will meet service-levels without money being wasted on over-provisioning due to unbalanced server load. This approach may be of use to storage systems, or to control agents seeking to make more informed decisions about their target system.

The contribution of this thesis is an investigation of the control of elastic storage where uniform load is not assumed. A control mechanism is presented which determines how to position data in the storage system based on demand or performance associated with the stored data.

The greedy-heuristic approach to solving the bin-packing problem this presents was previously published by [2]. This work reproduces this part of their work in relation to a different storage system, the Voldemort eventually-consistent key-value store.

Results

Our results, presented in Section 6, indicate that fine-grained workload monitoring does indeed improve controller responsiveness by decreasing the amount of data which needs to be moved to improve performance, and may reduce the service's consumed resources relative to a naïve uniform-load assuming controller. Although incomplete and inconclusive, the results are promising, and we suggest avenues for further experimentation which time did not permit.

Context

This work was carried out under the guidance of Ahmad Al-Shishtawy and Associate Professor Vladimir Vlassov, who have related publications and ongoing research on self-management and automatic control for storage and other services [3][4].

Chapter 2

Cloud storage characteristics

This section introduces concepts relevant to the use and management of elastic cloud storage services.

2.1 Cloud services: compute, content-delivery, storage

We may begin by differentiating storage from other cloud services such as compute and content delivery networks. Compute nodes are stateless, and as such can be made to scale horizontally. This can be seen in production with Amazon EC2, and the Google AppEngine and Heroku PaaS platforms. It is also addressed in [7] which presents a feedback controller for elastically scaling the size of an Apache Tomcat server cluster. Issues largely relate to available APIs, and to deriving appropriate feedback control logic in initial works. Such is helpful, but not sufficient, for controlling cloud storage systems.

CDN content delivery systems are also storage of a sort, but are quite different from the storage systems we consider here in that they are write-only and not cluster-local. Instead their emphasis is on immutable publication, the dissemination of a specific stored object to clients with good data locality, to save network bandwidth and reduce response times. They act as a caching, rather than state persistence service. That said, some of these concepts may still be transferable for provisioning the caching layer's consumed resources.

The key difference setting apart storage, as noted by [2], is the location of specific data items on specific servers: not all servers can service a request to read or write that item. Issues such as replication, consistency, query routing and performance constraints make controlling storage a tricky proposition.

With the rise of web-scale computing, championed by the likes of Amazon, Google, Facebook and Twitter, system engineers have increasingly found that traditional database systems are difficult to use with this new form of traffic. Rather than transaction processing with the ACID guarantees, focus has shifted to user-experience in terms of response-time and service availability. Furthermore, the social web causes data to interact in quite different ways: a user is

no longer interested in only their own data, leading to a natural partitioning scheme where their data is all co-located and easy to perform arbitrary SQL join operations and the like, but rather they want to see other user's data, Facebook being an example. with applications querying data in quite different ways from before many users and quite different data consistency [5] provides a survey of cloud storage systems; additionally a more hands-on reader may find [6] of interest.

2.2 Quality of Service

In order to keep users happy with a system it is typical to place real-time constraints on its performance: SLAs, comprising SLOs. Menasce's article [8] provides a good introduction to quality of service concepts. For more on user satisfaction see [9] and [10]. User experience is also a motivation for Amazon's Dynamo work [11], where choice of percentile as the SLO metric is driven by a desire to provide a good service to (almost) all customers. Google and Microsoft have presented jointly on the impact of response time on user behaviour for their search engines [12].

2.3 Elasticity

In much the same way as elasticity provides tangible business benefits to a consumer such as cost-cutting, it is probable that business policy rather than technical decisions will drive controller behaviour. For example, a service provider might not require their controller to always follow the demand curve; rather only being interested in provisioning for spikes, or cost savings associated with scaling down according to diurnal usage patterns. Whether and to what extent service level violations are acceptable to the client and service provider are policy decisions, and it may be useful to consider this before designing controllers which assume certain behaviours are always desirable. For example, a service provider utilising additional resources to avoid all violations will have higher operating overheads than one which is more sloppy, allows some violations, but reduces its server count and saves money. This should then be seen in the prices and SLAs they offer.

2.4 Common traffic patterns

Application workloads should be considered case-by-case, though general patterns have emerged for web applications which are useful in controller design and evaluation. We will consider three workload models, linear change is assumed to represent ordinary growth or decline in popularity; exponential

growth is seen in the so-called "flash-crowd" effect of short-term surges in demand; and cyclic workload variations are captured by the "diurnal pattern" of daytime and night-time usage.

2.4.1 Diurnal patterns: predictable cyclic usage

A diurnal traffic pattern is one where daytime usage is regularly greater than that at night. This pattern has been found in measurement papers to apply to particular web applications: Duarte et al found it in "blogosphere" activity [13], and Veloso et al [14] for live streaming video. It has been seen, however, that the global nature of the internet can skew this expected pattern, as in the case of DNS servers [15], so should be assumed carefully, and preferably measured.

As this pattern is by definition cyclic, it is possible to predict and adjust provisioning for the day and night periods based on expected demand for the respective diurnal phases. Given a simple day and night pattern, the system may simply choose to switch between two known configurations, one for daytime and one for night-time. However, while the shape of the change in demand is predictable, the amount of demand in the respective phases may change over time, requiring updating of the two configurations.

2.4.2 Flash crowds: viral popularity growth

A common pattern since the advent of online social media, such as Twitter and Facebook, is that successful new websites or applications may experience exponential growth in popularity over a short period of time. This is not a new phenomenon, previously it was known as the "Slashdot effect", taking its name from the popular technology news site, where websites accustomed to small numbers of visitors, in many cases running on a single web-server, would receive a massive surge in demand following being linked to in an article, prompting thousands of Slashdot readers to visit the site while the news item is fresh. Web 2.0's focus on user content has resulted in the creation of more websites providing a similar service to Slashdot, and so the effect is now if anything more widespread.

While a detailed analysis of the business and monetisation implications of this behaviour are best left to business analysts, it seems to be widely assumed that this is desirable behaviour for new internet companies. At worst, as might be considered by sites hoping to maintain steady and reliable operation, it might be seen as analogous to a natural disaster: something undesirable but necessary to have contingencies for during infrastructure design or planning.

This traffic pattern is problematic as it sees server workload quickly rise from their typical range, for which they will likely have been optimised to keep operating costs low, to a higher order of magnitude of traffic which is beyond their capacity to service as desired or dictated by their SLOs. The consequences of failing to provide good service to users was discussed in section 2.2, though

it may be further noted that in this case many of the visitors are new users, who may never return if the page does not load and work well enough capture their interest in the site's offered product or service on this first visit.

2.5 Data partitioning and replication

Storage systems are expected to provide a number of guarantees regarding the data they hold. These include durability: the notion of not losing data due to system failures, and availability: the notion of data being available within a bounded response time for some proportion of all data requests.

Both durability and availability are addressed in storage systems through replication, the practice of holding identical copies of stored data on multiple nodes. This introduces hardware redundancy to the system and making it far less likely that hardware failure will cause data-loss, and enables load-balancing between replicas to improve availability.

Finite data-capacity is a constraining factor on replication; each storage has a finitely large hard disk or system memory to store data. If each node holds all of the system's data, then the storage system inherits this limited data-capacity. While providing a simple replication and load-balancing model, this upper bound on stored data is not desirable. Data partitioning solves this problem by dividing the space of possible stored items into subsets, or partitions. Each stored item is deterministically assigned to a partition in a system-specific fashion. Examples include partitioning by database table, by primary-key or key range, or by segment of a ring onto which values of a hash function are mapped, such as consistent hashing [16, 11]. Having divided, or partitioned, stored files, each storage node will hold one or more data partitions (subsets). Queries for this data are then routed to the subset of storage nodes holding replicas of this partition of data.

Fundamental results in distributed computing unfortunately complicate the replication of data in a distributed storage system, with such issues as unreliable failure-detection, distributed consensus for update operations, and distributed data consistency. We assume the reader is already familiar with these topics, else suggest [17, 18, 19] as reference texts.

The consistency model of a given storage system is important to consider here, as it places theoretical rather than API limitations on possible actuation mechanisms. In particular consider stores offering strong consistency: here an increase in replication degree for a partition would actually worsen its performance (as may be verified by experimenting with the Paxos [20] algorithm, or pursuing its literature). This contrasts with eventually-consistent key-value stores, where reads from different replicas may produce different results, or different versions, depending on the adopted concurrency control mechanism ([21]) , but offer vastly superior performance.

In this work, we consider partitioning and replication as a means of achiev-

ing greater availability, and take for granted durability and consistency, assuming that they will be addressed by the storage system, and that aside from different systems posing different constraints on the controller, the approach we present here is not in conflict with these points; indeed it may provide a means of improving performance even for strongly consistent stores. Due to time constraints these points have not been pursued further in this work.

2.6 Data migration

Data migration is an expensive but necessary part of elastic storage control. While we want to be able to change the number of nodes, doing so means copying data from other nodes which are already serving user requests, and perhaps other disturbances to the cluster while migrating or repartitioning. This additional work hampers the system's ability to serve its clients in the short-term, but once complete its performance should be improved. Finding an efficient manner to determine which state to transfer, and trading its copy duration for additional server workload, is a key issue in storage controller design.

Different systems will require different kinds of data migration. For example, a storage system where each node holds the entire data-set will be able to have new nodes access data in parallel from all nodes, spreading the load evenly and not hurting performance for some users more than others. This is atypical however; partitioned data is more common, and will result in a subset of nodes being able to provide the data a new node will host. If these nodes are overloaded, the additional work will be unwelcome, but may be necessary to improve the performance of those files in the long-term.

An additional concern is which data to transfer. Systems making use of consistent hashing may move seemingly arbitrary data when the cluster's nodes change. Other systems, such as 3.2, target the items generating heavy load for the system, and move or replicate those items only. By reducing the amount of data transferred they are able to achieve big performance improvements in a small time at a lower performance overhead. They must still worry about how fast to copy the files however, as it will disturb the system's performance.

Aqueduct [22] is a control system for SLA aware data migration in live production systems. It throttles data transfer rates with a feedback controller, striking a balance between transfer duration and SLO impact. While appealing, it is not clear how compatible it would be with this problem, as the target system may already be violating SLO, making the controller's task to escape this state and return to safety as soon as possible, but without doing excessive additional damage to SLA. In this case we should optimise the total SLO violations, which will involve the duration of our SLO-violation period and the worsening of the number of SLO violations brought about by our data transfer. Lim's paper denotes as further work the investigation of rebalancing controller policy,

though [23] Figure 9 provides a measurement comparison of the extremes and one static coefficient based compromise. Flash-crowds are an example of traffic which could lead to this situation.

Diurnal traffic patterns do not require a fast response, so less bandwidth can be allocated, but it is sensible to still complete rebalancing as soon as possible without causing violations, so that the controller remains responsive, not blocked by long duration rebalancing operations. Considering both flash-crowd traffic changes and diurnal patterns, it appears that a desirable goal for the rebalancing controller is to minimise both the number of SLO violations, and duration of rebalancing.

Chapter 3

Related work

In this section we review two prior systems addressing the automatic control of elastic cloud storage, (3.1, 3.2), and discuss relevant papers on machine learning and architectural approaches to such a system. Our focus here is the published systems, as their ideas and limitations provide the motivation and inspiration for our prototype partition-aware rebalancing mechanism, presented in section 5. In particular the approach taken by the two systems to workload partitioning, monitoring, and load balancing should be carefully weighed.

3.1 Feedback control of HDFS in the cloud

In [23] Lim, Babu and Chase describe an integral controller for 3-tier web applications deployed on IaaS platforms, where the number of provisioned nodes is minimised to save money, but SLO violations due to under-provisioning should be avoided. Their work focuses on the storage tier, addressing the issues of discrete actuators, actuator lag, and measurement noise generated by actuation.

Response time is the reference input or desired output, and is obtained by transducing CPU utilisation, which was found to correlate with response time for this application. Its beneficial measurement properties make it preferable to response time as a measured output for the system – it's easy to measure, and has a relatively stable signal. The controller takes sensor readings by RPC to the HDFS leader (Namenode), which collates views of the cluster's CPU utilisation from readings piggybacked onto the storage nodes' heartbeat signals used in failure detection.

While the controller makes several assumptions about the target system, the most important is that of load-balancing and replica management: the controller allocates resources according to observed demand, and the target system is responsible for putting them to good use. Uniform load is not assumed by the controller, but is adopted for the prototype evaluation due to the inability of the HDFS rebalancing mechanism used to balance load across files.

Actuation is on two variables: the size of the cluster, and a bandwidth lim-

iter on HDFS's rebalancing mechanism. Each of these actuation points has its own controller. They operate concurrently, with a mutual exclusion relationship presented in the paper as a state-machine. The cluster size controller waits for its previous transaction to complete (finish rebalancing) before making further control decisions. This approach is taken to prevent oscillation due to actuator lag and sensor noise: the state transfer required to change cluster size introduces additional system load (sensor noise), and additionally the controller must consider pending resizing operations before requesting further changes (actuator lag).

Controlling cluster size

Cluster size is controlled by an integral controller utilising dynamic hysteresis and a workload model connecting sensed CPU utilisation to system response times. Hysteresis is used to address the issue of cluster size changes having fixed amounts – we cannot add half of a server. The paper's extension of this, "proportional thresholding" (3.1 of [23]), addresses the disturbance input stemming from changes in cluster size. That is, the measured output is relative to a single node, while the required control input at any time is a function of the control error and the cluster size, since adding a single node to clusters of size 10 and 1000 will see quite different reductions in per-node CPU utilisation. This highlights an additional concern in selecting measurement attributes, especially when scaling to or collecting for a single node in distributed systems rather than considering more abstract system-wide work units and capacities.

Controlling state transfer rate

The second controller allocates bandwidth to HDFS for rebalancing its data layout amongst cluster nodes. It trades actuator lag (the duration of rebalancing) for service disruption (the deterioration of response time induced by the rebalancing work). Rebalancing quickly adds significant load to the system, worsening SLO violations. Slow transfers, as in Dynamo [11], increases actuator lag, making the controller less responsive, and diminishes or removes its ability to respond to fast changes in traffic such as flash-crowds or short-duration spikes. This is discussed further in section 2.6.

3.2 The SCADS Director

The Berkeley SCADS system [24] is a "Web 2.0" storage system with a number of novel design goals, aiming to ease the burden of optimisation on the developer as an application scales. For our purposes it can be considered an eventually-consistent key-value store, as presented in [2]. The SCADS Director is an experimental elastic storage controller which manages both the provisioning of resources and the layout of data (partitioning) for load balancing.

The work focuses on upper-percentile response time guarantees, which can be seen in some of its design choices where expensive options such as additional replication are taken. However, as stated these are not compulsory, can be set by policy, and the paper contributes a number of ideas we found useful in this study.

Measurement by performance model

Upper-percentile response-time has a high variance, making it unsuitable for use as a measured input in control. Instead, the controller observes the system's workload, and uses a performance model of the storage system to detect when a workload is likely to violate SLOs. Based on this estimation of "overloaded" and "underloaded" servers, an action policy set is executed against the servers to rebalance load by migrating data to underloaded or new servers, and remove unused servers.

Replicating for upper-percentile latency

An additional aspect of this work is the focus on upper-percentile SLOs, which motivates two expensive replication decisions. Each user request is performed by multiple nodes (without quorum) so that if one server experiences an upper-percentile causing glitch in its performance, the other replica is likely to still return a result in good time. Furthermore, nodes are provisioned but not utilised by the storage system, idling until the controller decides to include them in the storage group. This undoubtedly speeds up adding nodes to the cluster, making the controller more responsive and better able to prevent violations, but as the idle server must still be rented, it is responsiveness at a price. One might also wonder why not utilise the idle node, and make the controller more sensitive to changes in workload level. Whether either model is effective in responding to flash-crowd spikes is something akin to preparing for a lightning strike: it depends on the performance model of the storage service, and on the magnitude of the spike.

Controlling data partitioning

Data migration is harmful to performance, as discussed in 3.1. In SCADS the controller adopts the responsibility of rebalancing the storage system's data layout, and it does so by copying as little as possible. By monitoring the demand for particular file partitions the controller is able to identify popular partitions and increase their replication, or move them to empty servers.

This migration entails two complications: optimising the location of data on the fewest server resources is computationally complex, in fact mapping to the classic NP-hard bin-packing problem, and once a plan has somehow been devised it may need to be changed due to changes in system workload.

The optimisation problem is addressed with a greedy heuristic approach, through the Action policy set, which maintains a reasonably efficient data layout while keeping the amount of data transferred small (general approximation algorithms might completely change the data layout; having an existing position for items in bins and wanting to move as few as possible is an additional complication).

Changes in workload are addressed by planning migration actions, queueing them, then executing them in order until changes are observed and a new plan is devised. The presented policy set schedules scale-up actions before scale-down actions, meaning that the removal of under-loaded servers may be delayed until later when spikes in demand are observed for certain files, requiring quick replication or relocation.

In summary, key concepts from this work were taking control of data layout, reducing the amount of migrated data through partitioning or grouping stored objects, the framing of replica layout as a bin-packing optimisation problem, the use of Action policy sets rather than feedback control, and the use of a performance model to avoid measurement noise.

3.3 State-space storage control

[3] presents the difficulties in modelling a target system, or its "identification", for making control decisions based on past sensor input and control output. It explains that building analytical models for complex computer systems, such as storage, is prohibitively difficult, and that past work has largely involved the black-box empirical approach of measuring the real system's response in various configuration states. Empirically modelling a system can also be difficult, for high-dimensionality complex systems, though approaches to this will be referred to in 3.4.

The contributions of the paper are a cloud storage control simulator, and an evaluation of the state-space control model in this simulator. State-space control makes use of regression techniques to determine controller parameters, such as gain, from empirical system data. We believe this to bear similarities with machine learning approaches, which we discuss next.

3.4 Machine and Reinforcement Learning

Bodík et al argued in [1] that while machine learning is a sound approach to self-management, existing systems had not made use of the necessary techniques to make controllers which could handle real applications.

In another paper Bodík [25] addresses the issue of model learning in live production systems. It suggests the training and refinement of performance models based on real data at real scale, rather than training with static data or replaying traces. A contribution of the work is a controller which achieves this

without exposing the system to excessive SLA violations. It is written with web 2.0 data-centre (cloud) systems in mind, though the ideas are of general interest. An interesting observation made by the paper is that small-scale systems are insufficient for training a resource controller, as bottlenecks and workloads behave differently when scaling crosses to new orders-of-magnitude.

Vengerov [26] describes a reinforcement learning controller managing the positioning of files in hierarchical storage, in the sense of making decisions on caching and cache-eviction at multiple tiers (hard disk, ram). Interesting insights are presented on storage system usage and workloads, though they should be taken cautiously, as some are not cited or substantiated by measurement, and may be derived from systems differing from those in production today. Two clear contributions are made: a framework for applying policy-based control to hierarchical file storage, and a reinforcement learning algorithm for optimising policy coefficients.

While we do not pursue these ideas further in this work, machine learning appears to be a topic to watch in relation to autonomic control. We believe our own work could adopt system identification techniques discussed herein, or could provide one actuation mechanism controlled by such a learning control agent.

3.5 Load shedding

Typically peer-to-peer storage research has assumed uniform load on data items as in [27], though there has been some investigation of applying distributed system load shedding techniques in DHTs [28] [29].

3.6 Architecture, methodology, and surveys

Three works in particular were found useful during the early stages of this work: Kramer and Magee's architecture paper [30], Tesauro's multi-agent systems discussion of autonomic computing [31], and Al-Shishtawy's methodology for self-management paper [32]. Between them an overview of system control can be formed, unifying approaches from across computer science rather than focussing on the application of control theory.

Additionally, the YCSB paper [5] introduces a measurement framework for cloud storage systems, as well as providing a good overview of available storage systems and their design-space.

Chapter 4

Elements of Elastic Storage Control

This section introduces relevant concepts from control theory and distributed computing pertinent to the automatic control of cloud storage systems. Control Systems Engineering is a rich field in its own right, and this does not aim to provide a comprehensive introduction. Instead we refer the reader to Hellerstein's text on control for computer systems [33].

4.1 Control theory terminology

When we discuss control, we refer to the observation and manipulation of a system's state to maintain a desired behaviour. Examples of controllers include cruise control for cars, temperature regulators for ovens, and thermostats for heating systems. In each of these systems there are clear things to observe, such as velocity or temperature, and to change when those observations differ from what is desired, such as engine or heating element power. There are three main components to determine when designing a controller: how to sense the system's state, how to actuate change in the system, and how to derive appropriate changes to the system from the sensor data.

We begin by introducing some terminology, based on that of Hellerstein et al [33].

Target system
: The system or device managed by the *controller*.

Controller
: The device we are designing, which determines how to set the *control input* to achieve the desired *measured output*.

Measured output
: A measurable characteristic of the target system, such as CPU utilisation

or response time. Relates to management goals via *control error*. Also "sensor input"

Desired output
: The desired value of *measured output*, for example 50ms response-time or 60% CPU load. Also termed "reference input" or "setpoint". Relates to *target system* behaviour via *control error*.

Control error
: The *measured output*'s offset from the *desired output* (converted by the *transducer*, if applicable).

Control input
: A dynamically adjustable *target system* parameter which affects its *measured output*. For example the number of server replicas in acting as a website's front-end.

Transducer
: A mechanism for converting the *measured output* to a form comparable with the *desired output*. For example, a controller might enforce response time by measuring CPU load. This notion of correlated indirect measurement is discussed elsewhere.

Measurement noise
: Distorting effects on the measured output, also termed "sensor noise" or "noise input".

Disturbance input
: Changes affecting how the *control input* relates to (effects) the *measured output*.

4.2 Issues in system measurement

Being able to assess the current state of the system is essential to any controller. However, measurement is no simple task. Aside from the typical concerns of selecting what to measure, how to instrument it, how measurement will interfere with behaviour, noise, precision and resolution.

In order to control a target system we must obtain a measured output to base control decisions on. Measurement is a complicated science in its own right, further complicated here by distributed systems properties. This section briefly discusses some of these concerns and how they might affect controller design.

4.2.1 Selecting a metric

Here it is necessary to consider which metrics can be obtained from the target system or its components, an appropriate collation approach, and the effects of distribution and scale on these measurements. As a system grows some measurements will become too expensive, or their results may suffer from increasing error and noise due to delays or failures.

A system's measured output may be something obvious, such as response time, or it may be indirectly measured from another property such as request count and transduced to a response time estimate using a system performance model.

This transduced measurement approach was adopted in the SCADS Director (3.2) due to the high variance of sampled 99th percentile response time, making stable control problematic. Signal filtering could be applicable here, but may slow a controller's response to sudden change such as flash crowds (2.4.2).

In collecting measurements it is likely that statistical aggregates of many samples will be used to represent the current state of the system, or its constituent parts. The scalability of these measurements varies, for example cumulative mean appears to be easier to collect at large scale than percentile readings which require histograms rather than a single figure to be stored.

4.2.2 Granularity

An important consideration in measuring a large system's behaviour is how much information we would like, or need, and how much we are willing to pay for it in terms of performance. Below we present a number of granularities, or resolutions, at which we might be interested in the performance and behaviour of a cloud storage system.

Granularity should not be confused with precision, which is a complex topic often omitted from systems research, also in this study. [34] is suggested for an introduction to measurement error analysis.

System load

A simple count of get and put requests made to the storage service as a whole. This could be useful if the performance model is simple, as in [2]. However, it would not capture the distribution of those requests across the nodes in the cluster; uniform load distribution is assumed, and may not be the case.

Node load

Here the access to each storage node would be tracked, to determine when the cluster contains nodes which are overloaded and violating their SLOs, or which have low utilisation and are candidates for removal. However, it does

not provide information about the stored objects or partitions responsible for their experienced workload, for example a very frequently accessed item.

Stored object load

By monitoring the demand for individual stored objects, the controller can make informed decisions about the distribution of data on the storage cluster, and which individual items are hot and require replication. However, as the number of files in the system rises, this information can become expensive to collect.

Partition load

By monitoring the access to data partitions (key ranges or arbitrary groups of objects) the controller and system can reduce the overhead of monitoring and reasoning about the system's data access.

Two examples are the reduction in stored measurement results from 1-per-file to 1-per-partition, and a smaller number of items to organise when bin-packing. However, this does mean that precision in identifying hot stored objects is lost, meaning that more data than necessary will be replicated.

4.2.3 Measurement locality and distribution

Having obtained measurements at individual nodes, they may be collated at a central point, or shared between nodes in a peer-to-peer fashion.

Viewing global system state is problematic, and distributed aggregates and analytics for example are a topic of research in their own right. Other related topics include distributed state, deadlock, and failure detection.

It may be that one observation is sufficient, for example we might take a single node's performance as representative for the whole cluster if uniform loading is assumed. This may be improved by looking at several nodes, enabling averaging to remove noise from the readings. Care should be taken in using a single aggregate however, as significant information may be lost, such as demand spikes at a single node.

Central measurement collation

A typical measurement model is to have a central node responsible for collating and interpreting measurements from the nodes in a system. As with many distributed computing problems this is a sensible starting point, but introduces a central point of failure and bottleneck for scaling which must be addressed later.

An advantage of this approach is that the central node can establish a canonical global snapshot, which it may forward to control logic which makes decisions about the system.

Implementations range from ad-hoc central database connections, to communication systems such as Chukwa [35] and to high-end stream-processing solutions. Examples of this approach are seen in the control systems discussed in Sections 3.2 and 3.1, which respectively make use of a MySQL database and the group leader as central collection points.

As the cluster grows it will become prohibitively expensive to track metrics such as average CPU utilisation across the cluster, essentially a global system view - one of the hard problems in distributed computing. While it remains possible to obtain measurements from the cluster, decentralising them, or taking partial rather than complete system views, may change the stability and semantics of the sensor readings when compared with a centralised reading for a smaller cluster.

Two clear opportunities to make use of this in storage control are a front-end load balancer coerced into data collection, and a dedicated measurement component. A dedicated central measurement component would collect readings from measurement agents in a push or pull fashion, requiring either group membership or coordination of measurement component location. Where the storage is accessed through a front-end load balancer, it will be possible to obtain information about the client requests being made to the system. This could be a simple request count, or a detailed analysis of the operations and accessed data.

Centralised control is a simple and often adopted distributed systems architecture. Its scalability is limited, and it presents a single-point-of-failure, but it is also a simple and pragmatic starting point, and often sufficient for production systems when carefully used.

System front-end metrics

Having already identified a front-end load balancer as a possible collection point, we might ask why not measure the system's utilisation at the front-end, rather than instrumenting individual storage nodes. Indeed, this may be effective for a number of metrics, and should not be ruled out, though will place additional load on a system component which should operate very quickly. Given that load balancers may also be replicated, this does not rule out distributed measurement concerns entirely.

Real-time constraints and meaning of global measurement

While there will be a delay between any measurement and its use, these are often imperceptibly small and ignorable; in distributed settings however there is concurrency of like measurements at different nodes to consider in addition to the delay in their collection and use. When looking at an assembled collection of measurements from the system, it is highly improbable (at best) that they were taken at the same moment in time.

This does not become much of a problem when a system is taking measurements in the order of several seconds or greater; naive best-effort measurements will work well enough. However, as the time period for measuring reduces, the demands on the freshness and consistency will increase, and greater care need be taken in instrumenting, collecting, and interpreting the readings. For example, does it make any sense to measure 1ns intervals if results are to be collated over an unstable 100ms network connection?

4.3 Control decision models

Having obtained sensor data indicative of the target system's current state, there are numerous ways to decide whether and how to change its behaviour. These range from simple conditional logic statements, to simple or complex mathematical models, economic models, and AI techniques.

4.3.1 Policy control

To a computer programmer, this is the most obvious approach to solving the control problem. Conditional statements will be used to set conditions for the execution of control actions. Many such decisions may need to be enumerated, and it is unlikely that all situations will be covered [36].

4.3.2 Goal based control

In this model a controller is told to maintain a certain system state, but not how to achieve it. Deriving plans of action to maintain certain system constraints are the task of the controller. This is the level at which we would intuitively place SLA goals, though when considering the financial implications of violations, we lead in to a more general notion of utility.

4.3.3 Utility functions

The most general control objective is a utility function, the notion of assigning financial value to various system states, and assigning controllers with the task of maximising the utility of their target systems. To achieve this it may make its own decisions regarding both goals, and actions taken. However, utility can be difficult to assign in a meaningful fashion to systems, making the adoption of this model problematic.

4.3.4 Determining a suitable response value

A critical issue in control is determining how much to change the control input by. In some systems we might have a readily available equation to determine the control input, whether from the control error or some other measurement.

This might be the case for well-studied physical phenomena, power electronics, and computer systems with appropriate analytical models, such as those using queuing theory to connect response-time, throughput and queue-size.

In other systems, particularly complex computer and software systems, we may find our application's performance model to be complex, analytical approaches to be insufficiently accurate or overly complex and brittle, and require another approach.

Machine learning, reinforcement learning in particular, offers techniques for inducing function approximations from observed data, and has been applied and argued for [1][25][26] in storage system applications.

4.3.5 Three-layer control architecture

In [30] Kramer and Magee propose an architectural model for self-management inspired by Gat's 3-layer robotic control architecture [37]. Their aim is to bring benefits from developments in AI and robotics to the self-managing systems domain. The model identifies three layers of activity in a controller: control, sequencing, and deliberation.

These layers provide a familiar abstraction, enabling lower layers to be concerned about system specifics and implementation details, and higher layers to be concerned with more general concepts such as goals and constraints on emergent system behaviour. The control layer is responsible with system interactions: sensing and actuation. Sequencing receives sensor data and sends control signals back to the control layer's actuators, as directed by pre-compiled plans. There is a further interaction between the control and deliberation layers, where the deliberation layer receives system state, and sends revised control plans to the sequencing layer.

Here plans may be some functional input for the sequencer, such as an optimal layout it should reconfigure the system to achieve, or could comprise reconfiguration or replacement of the sequencer, as for new control loop coefficients or new action policy sets.

Examples include route planning, replica location optimisation, buffer region sizes for control or hysteresis, or the switch from one set of action control policies to another more suited to the current state. The retraining and refinement of performance and decision models would also fit in this layer; training from data may be expensive, but here we see that the sequence layer may continue to operate with the previously provided models until reconfigured by the deliberation layer, once they are ready.

The aim of this approach is to help in reasoning and understanding a controller's interactions with the system under control, and its own self-updating mechanisms necessary to provide autonomous control for a dynamic system. It is believed that this will help structure the controller in a modular fashion, easing design and implementation.

An example of this model in storage control can be found in the migration component of the SCADS Director. Their controller's heuristics for replica movement can be considered as the deliberation layer, the action executor its sequencing layer, and SCADS and EC2 interfaces encapsulated by its control layer. In this case there is a clear similarity between a robot driving from A to B and encountering a change in the environment which requires a new plan, and the partitioning migration plan, which is expected to complete a number of steps then be replaced by a new sequence of actions, particularly in the case that a significant workload change occurs: a flash-crowd is to the storage cluster what a rock falling from the sky might be to the exploring robot.

4.4 Actuation in elastic storage

Having identified that our system requires change to maintain its good behaviour, we must take actions effecting that change.

In section 4.3 we outlined models for making control decisions; here we will focus on available actuators in storage systems, and confounding problems they may present.

4.4.1 Number of nodes

Add nodes to the cluster when system utilisation is high, remove nodes when utilisation is low. This is the most general actuator, being the fundamental unit of control in horizontal scalability.

4.4.2 Data layout

Stored objects (or partitions) may be moved between storage nodes, or to new nodes, by the controller, to achieve a layout with optimal node utilisation. Further constraints include the total amount of of data to be stored (especially for in-memory stores), and moving as few data as possible to achieve the desired layout. Figure 4.1 illustrates rebalancing data partitions, or key ranges, between an overloaded and underloaded server.

4.4.3 Data repartitioning

Assuming that the system's data is partitioned, either in arbitrary bins or keyspace ranges, the controller could repartition the system data. When working on partitions rather than individual stored objects, the controller may need to repartition the stored data.

A situation where this would be useful is upon discovering that a partition is receiving so much demand that it cannot be held within a single server. If demand is for a single key, then only replication can improve its performance, though repartitioning could isolate the hot key from other keys, allowing the

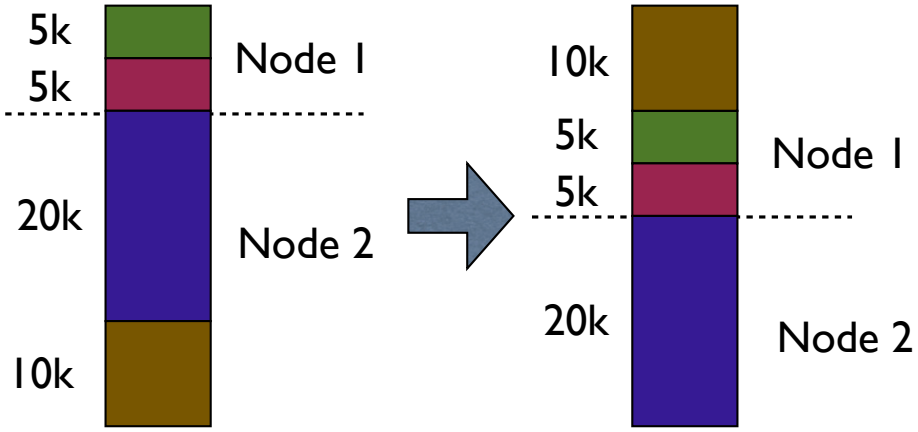

Figure 4.1: Rebalancing non-uniformly distributed partition workload between two storage nodes

performance of other keys to improve by moving them to other servers. In the case that the partition's load is spread across multiple keys performance can be improved by cutting the partition into smaller pieces and sending them to distinct servers.

One desirable property for the system's data partitions is that popular stored objects occupy small partitions, to reduce the cost of replication. A contradictory requirement is that the system have few partitions, to keep associated overheads tractable.

4.4.4 Slow data migration decomposition and sequencing

Storage control is reliant on network state transfer, an inherently slow process for large data. In order to control with a slow actuator we attempt to invoke it in such a way that it is interruptible in case we need to revise our control decisions due to a change in workload. In the absence of an abortable transfer mechanism, the transfer can be split into actions and serialised (sequenced) by the controller.

In order to decide how to sequence migration actions, the controller faces four optimisation tasks: maximise performance, minimise servers, minimise data transfer, and maximise effect/time of the sequence.

Maximising performance, minimising servers: bin packing

Maximising performance and minimising servers is simply the bin-packing problem, ensuring server workload is well-balance at the granularity of data partitions.

Minimising data-transfer

Minimising data transfer reduces disruption to the service. We might find a new layout that removes one server, but if lots of partitions need to be moved to reach it, we may decide it is not worthwhile due to the service level disruption incurred. Furthermore it is the constraint that all bins in the bin-packing problem are not equivalent, having found a layout, it is necessary to match bins to existing server states, minimising the overall distance (difference) from their current state to the planned layout, where this difference maps to required data transfer.

Maximising plan time-effectiveness

Having identified a target layout, including storage node allocations, we want to decide how to order data transfer actions so that performance gains are achieved close to the start of the process. This is both to make the controller responsive to step increases in workload, and because workload may change, not undoing the worth of our work, but makes it less important. For example, a traffic surge to one key might arrive while we are rebalancing a slightly overloaded server.

This extra planning is an additional computationally hard problem. However, if we take any route without considering the intermediary steps we may transition between many less optimal data layouts, worsening performance further, before reaching a more optimal one. This is additionally problematic if the controller needs to abort and re-plan the repartitioning (cf. Gat 3-layer's deliberation layer), as the repartitioning work done so will have worsened short-term performance and carried no long-term gain.

A simple approach is to prioritise scaling-out actions over scaling-down actions, and to act on the busiest data first. In the aforementioned case if we had moved a busy partition away from the overloaded server first, its performance situation would be resolved before we are forced to switch our efforts to dealing with the other, more severe, overloading elsewhere. If on the other hand we had been consolidating under-utilised nodes first, the slightly overloaded server's problem would not have been resolved.

Computational cost (complexity)

Having identified three optimisation problems, we might wonder how to solve them. Finding global optimal solutions to each with anytime online algorithms would be complicated and perhaps too slow, but is a direction worthy of further research with constraint programming.

A simpler approach is the SCADS Director's action scheduling approach (3.2) using an action policy set: the heuristic conditionals they present resemble greedy gradient descent; they may not find a global optimum number of servers, but they will improve performance, and are computationally cheap.

4.4.5 Unreliable actuators

In [23] we read that actuators do not always respond as expected. In this particular case, the HDFS rebalancer does not make good use of assigned rebalancing bandwidth greater than 3MB/s. Given the complexity of distributed storage systems, it is not unreasonable to expect that system actuation points may be unreliable, or operate correctly within a limited range. Whether appropriate, and whether the functionality can be fixed, should be considered case-by-case and verified by measurement.

We do not revisit this issue, as the complex actuator used in our evaluation did not present unexpected behaviour. However, we do suggest its optimisation as a future work, which could result in or expose existing unreliable behaviour.

Chapter 5

Building a partition-aware storage controller

This section documents the design of our partition-workload aware elastic storage controller, and its prototype implementation for controlling the Voldemort eventually-consistent key-value store. It begins by introducing Voldemort, then describes our controller in terms of measurement, control decisions, and actuation.

Figure 5.1 outlines our system. Conceptually, the Executor, Collector and Planner operate concurrently and communicate by message passing.

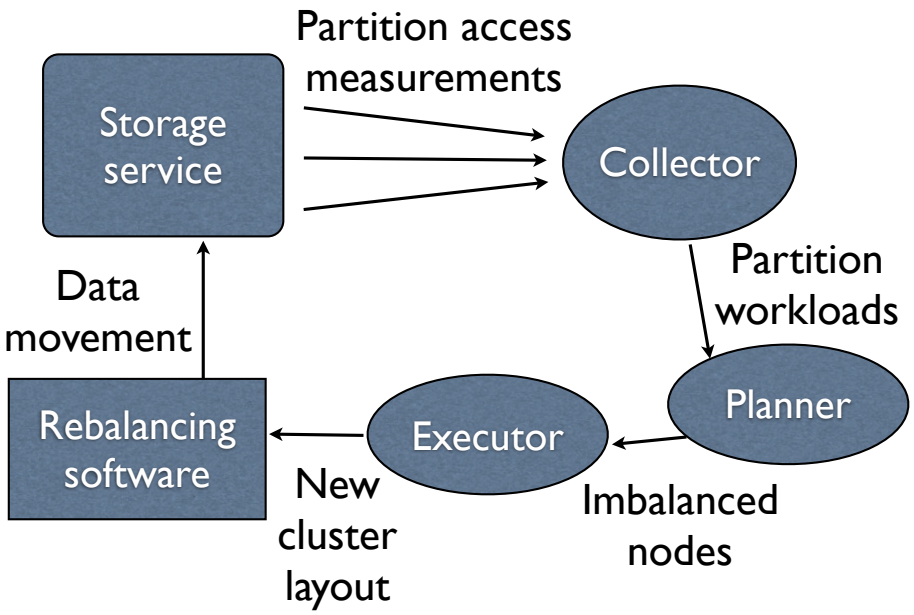

Figure 5.1: Cloud storage control prototype overview

We refer the reader to 4.3 for a more detailed discussion of control models. For this work, we adopt a simple action policy set controller approach, as our focus is on measurement and actuation techniques rather than system identification, modelling, or decision-making.

5.1 Storage service: Voldemort eventually consistent key-value store

Voldemort is an open-source implementation of Amazon's Dynamo eventually-consistent key-value store. It is produced and used by LinkedIn, an online social network, to serve data with performance and efficiency requirements beyond the reach of conventional database systems.

As the topic of this work is not specific to this one storage system, rather than presenting it in detail we refer the interested reader to the Dynamo paper [11], and the project website[38][1].

Our choice of Voldemort as a prototype component places important constraints on the abilities of our controller, as would be true with any store. The issues we consider most significant are discussed below.

5.1.1 Partitioning

The system's keyspace partitioning (discussed in 2.5) is fixed in advance, and cannot be changed (for each particular storage table). While their range is predetermined and fixed, the location of partitions may change during operation. Partitions can be moved, and Voldemort provides a "rebalancing" tool which intends to safely migrate data from one partitioning layout to another. The tool may either be given the current configuration, or retrieve it from the running cluster. The target layout is provided by the user as an XML file.

5.1.2 Replication

Replica locations cannot be fixed, so must be disabled else will incur additional load which we cannot control. This is unfortunate as it is a vastly unrealistic constraint to place on a distributed filesystem, sacrificing availability and durability of data. Partition layout optimisation with deterministic but non-configurable replication is deferred to future work. Here, instead, replication and quorum sizes are set to 1.

5.1.3 Data migration

As seen in the SCADS and HDFS controller papers, the migration of data is problematic for at least two reasons: it hurts performance, and it can block fur-

[1] `http://www.project-voldemort.com` accessed June 2012

ther controller action. Below we will consider three ideas: the effect of a single blocking repartitioning transaction on the controller, the impact of rollback or abort semantics, and the benefits, difficulties, and performance compromises of decomposing the repartitioning into smaller steps. The approach taken by SCADS was discussed in 3.2.

The following approaches were considered for the Voldemort rebalancing software. A first, simple approach is to determine the new partition layout to move to, and perform the repartitioning in one blocking uninterruptible operation. While the repartitioning is performed, the controller is unable to take further action.

It is interesting to consider aborting the repartitioning operation: in Voldemort it is made safe, with rollback semantics. Such functionality is good for maintaining consistency, but bad for our control situation, as we may have to wait for rollback to complete before we may try our new, revised plan, which may occur several times meaning we never get anywhere.

A more suitable approach is that taken by the SCADS Director, which is to determine the new partition layout it wants, then schedule small-step actions which will lead it to that desired layout. These steps are executed one at a time, each potentially blocking and non-interruptible as before, but of a much shorter duration. This is reminiscent of the 3-layer control model: a layer of higher-reasoning decides on a new partition layout, and provides the lower layers with a plan of how to get there. If the lower layers detect that the workload has changed significantly they will report back to the higher layer, asking it to revise the plan.

This does not mean to say that producing such a plan is trivial. The SCADS Director's model predictive control, or action policy set approach, sidesteps a number of optimisation issues, by jumping directly from layout to actions to take to maintain an optimal state. If we are instead to consider the best path from our current layout to a new optimal layout, with several intermediary steps, we should consider whether these intermediary steps are more optimal than the first layout, since we may abort and re-plan prior to reaching our optimal layout.

Another issue with decomposing the repartitioning is we may lose performance optimisations provided by the repartitioning system: Voldemort offers a number of parallel and concurrent transfer configuration parameters which may function less effectively if a small number of partitions are to be moved in a single step.

In the case that a vast change in workload is detected, the rebalancing process can be aborted (by cancelling remaining queued asynchronous jobs which enact repartitioning) and new control decisions made using the partially repartitioned, but consistent and operational, layout.

In this work we adopt single-step refinement of the cluster, without parallel transfers.

5.2 Sensing: measuring system performance

The Collector component polls registered instrumentation agents for sensor data, receiving a histogram of partitions and their request count since the last pull request. The specifics of this are described in 6.2.2.

Other metrics could be taken in the place of request counting. Indeed, this model holds for our simple in-memory get-only experimental situation, but when a full storage system is involved this will no longer be the case. Also, file size is taken as uniform across the store. This is unrealistic, and as network interfaces are found to be the current system bottleneck we believe it worthwhile to consider the required bandwidth, or file size, or transfer time as an alternative weight metric for partition workload. Having introduced unequal file sizes the bandwidth associated with each request will vary, and our simple request counting performance model will likely break down. A tricky issue here is not tracking which key in a partition is being accessed. Probabilistic techniques may enable the construction of an approximate analytical model, though state-space control and machine learning techniques may also provide interesting avenues of investigation. A simpler approach would be to simply monitor response times, though previous works (3.2) have found this to be unstable and discouraged its use.

5.3 Making control decisions

Having obtained measurements of the performance for particular partitions, we must decide how to change the storage system to improve its performance and efficiency.

Here we adopt the SCADS Director approach of queuing actions to perform until new information is received, at which point we re-plan and replace the command queue. Conceptually this fits the 3-layer model of sequencing and deliberation of 4.3.5. We sequence actions which will improve performance based on the current layout and usage, but we deliberate changes in workload and revise the action sequence.

5.3.1 Planning and deliberation: reacting to workload changes

The deliberation component of our controller is the most conceptually interesting, offering two divergent approaches. Two approaches present themselves for repartitioning the cluster to rebalance workload. Both make use of partition-workload measurement information, but their optimality and computation times are quite different.

Optimal partition arrangement by constraint-programming

By addressing the bin-packing problem head-on we can obtain the most optimal solution, and so the fewest number of servers, for our revised cluster. However, this form of computation can be time-consuming, slowing controller response, and is heavily dependent on the number of data partitions.

Additionally, as discussed in 4.4.4, once an optimal packing of partitions into bins has been devised, the bins must be fitted to servers in such a way that state transfer is minimised. This further optimisation problem exacerbates the initial concern about computational complexity and execution time. It may be that the proper approach is to modify the bin-packing constraints (such as in [39]) to find a solution resembling the current layout, but this is left for future work. It is suggested that Gecode, a C++ constraint programming framework, would be a good choice of tool to investigate this approach.

Action-set heuristic optimisation

For our prototype, we eschew the matter of computational cost by making direct use of search heuristics in our conditional logic to achieve greedy gradient-descent. This approach mimics that of the SCADS Director (3.2).

If a cluster node has a workload corresponding to an SLO violation, according to its performance model, the controller adds the node to a set of nodes to be considered for rebalancing, with an associated weight indicating how high its workload is. The performance model is currently a predetermined model shared by all nodes. Having considered all nodes, the weighted set of nodes requiring rebalancing is sorted and queued for rebalancing, heaviest first.

The deliberation layer here produces two node lists: overloaded and under-utilised nodes. These are passed to the execution agent, which uses its action policy set to determine how to move data partitions so that overloaded nodes can be relieved, and underloaded nodes given work or removed.

Nodes which are under-loaded according to the performance-model are added to a second set, of nodes which should be assigned files during rebalancing, or should have their data coalesced and some of them de-provisioned to save money. This concludes the sensing feature of the controller.

5.3.2 Executor: enacting sequenced plans

Having identified that a server needs to shed some of its workload, we must select partitions to move based on its workload and our performance model (presented in 6.3), and to where they should be moved. Our approach is simply to move the hottest partitions first, hoping to quickly improve performance with minimal data-transfer. Each operation is executed independently; we run the Voldemort rebalancing tool with single partition movement operations. Parallelisation of these operations, to achieve latency hiding, is left as future work.

Our sequencing agent, the Executor, plays the role of consumer in a Producer-Consumer relationship with the Planner. The Executor operates concurrently is a process, A separate process (thread) is responsible for consuming nodes from the rebalancing queue. This rebalancing process schedules one-step schema migrations to be processed by the Voldemort rebalancing tool, according to the following logic. This is achieved by generating a new cluster configuration file with this one change made, and executing the Voldemort rebalancing tool.

```
For the most overloaded node, n:
  p = most requested partition held by n
  if p's workload can be accommodated by an underutilised node
    then move p to the busiest node with sufficient capacity
    else move p to an empty node
  return updated n to the rebalancing queue if necessary
```

For small data partitions on unloaded servers this will take a few seconds. The latency of these operations in a loaded system will be seen during experimentation.

While we do not currently implement such behaviour, this queue makes it possible for the controller to abort the scheduled actions and replace them with a new plan, based on new measurements, without needing to delve into the details of Voldemort's rebalancing system. That said, the current system works by scheduling a series of asynchronous data movement tasks, much the same as we do here. We leave as a further work an investigation of whether this rebalancer can be optimised with our partitioned-workload information, or other ideas presented herein.

5.3.3 Shrinking the cluster

While we have addressed scaling-up to meet demand, we have not yet implemented scaling-down to save money. We may extend the described sequencing agent to remove unused nodes from the storage cluster.

We adopt a simple approach making use of the existing rebalancing thread. To improve a node's performance we remove some of its data. To remove a node from the cluster we must remove all of its data. As these are quite similar, we may re-use the existing rebalancing thread, by extending its logic to cover downsizing when it is not busy scaling up the cluster or adjusting its partition distribution to maintain SLOs. When the rebalancing thread finds that no nodes are overloaded it considers instead whether it may remove nodes from the cluster. In this case it looks at the under-loaded nodes data structure, selects the node with the least workload, and attempts to move its partitions to others. Once all of its data has been removed the node can be removed from the cluster.

5.4 Actuation: moving data

Our controller's actuation mechanism is moving data on the storage cluster. It produces a cluster configuration detailing which servers will hold which data partition, and sends this plan to Voldemort's rebalancing tool which moves partitions to match the new plan.

Voldemort provides a rebalancing tool, which takes a new cluster configuration file defining the member nodes and partitions they shall hold, and determines a plan of asynchronous jobs to safely transfer the partition and its data from the current node to its new location.

Currently an unused server is still running but allocated no partitions, so plays no part in serving requests. To enable real elasticity we should add and remove cluster servers based on their usage: remove when empty, add when the cluster is too small to provide a more optimal partition location plan. Rather than complicate our controller, we opt here to leave this management of cluster size for an additional controller, and rely on ours to redress imbalances in workload with the resources it sees currently available. It could request that another manager add additional resources, but delays in adding such resources mean that it must still wait for them to come online before using them in this form of partition location planning.

5.5 Implementation details: languages and communication protocol

The controller was implemented in Ruby (`http://ruby-lang.org`), with threads and shared-memory communication. Ruby's GIL (global interpreter lock) is problematic here, preventing parallel thread execution. JRuby (`http://jruby.org`) offers one solution to this problem, though reimplementation in Erlang or Java may be preferable to achieve greater scalability through parallel execution of the concurrent processes, and better interoperability support.

The storage service, Voldemort, is a Java application, as is the chosen load generator. Instrumentation was consequently carried out in Java, making use of available concurrency libraries.

Java to Ruby communication is performed by serialising data structures to YAML strings and transmitting them, UTF-8 encoded, over a TCP pipe.

At `https://github.com/nruth/controldemort` full source-code is available, including the instrumented Voldemort and YCSB, our controller, and assorted measurement scripts.

Chapter 6

Evaluating partition-workload aware storage control

In this section we present the experimental results for our partition-workload aware controller for the Voldemort storage system. We begin with a discussion of the chosen measurements, then describe our experimental method and results.

6.1 Measurement goals

In order to evaluate our control mechanism's effectiveness at load balancing we seek to capture the dynamic behaviour of the target system as it experiences changes in workload. In order to visualise this, we will record 99th percentile response time, and a representation of the workload the system is experiencing – in these results its completion rate. Behaviour here in its simplest form is response time, the key SLA metric.

Having integrated the controller and instrumented the target system, it should be operated under fair conditions to demonstrate that the controller does disrupt steady-state operation with irrational behaviour. A suitable experiment here would be to evaluate its behaviour under a generated workload with uniformly random key accesses. This workload could then be varied by either a step or steady increase in overall arrival rate to approximate the flash-crowd and diurnal traffic patterns respectively.

While further measurements to verify the controller's behaviour could be included, the prototype's purpose is to demonstrate the disadvantage of assuming key-load uniformity. As such, we instead seek evaluation of the controller under pathological workloads which demonstrate the difference. In particular, the case where overall load is constant, but shifts from a uniform distribution to being concentrated on on a subset of partitions.

6.2 Experimental method

This section briefly surveys measurement tools for cloud storage systems, the method adopted herein, and a second version of the controller devised for comparison with one assuming uniform load.

6.2.1 Measurement software

The main requirement of the chosen software is multiphasic workload generation. Without this, we cannot produce diurnal patterns or flashcrowd spikes as discussed in section 2.4. Three evaluation tools have been seen in closely related literature: the EStoreSim simulator [3], YCSB [5] and Cloudstone [40].

Cloudstone is a benchmark for deployment environments, to compare the performance of different cloud configurations and server stacks (including load balancing proxies and caching layers). While its interest is in measuring value-for-money, or cost-per-user, rather than throughput or latency, the specifics of SLA violations and dynamic workloads or traffic shapes are left for future work. Furthermore, while the stock benchmark utilises relational databases for persistence, in [23] section 4.1, we learn that their evaluation is based on a modified build of Cloudstone where HDFS has been added to the application. Cloudstone's chosen load generator, Faban, is of interest in its own right, as a means of providing probabilistic workloads with varying workload phases.

YCSB is a stress-testing benchmark for cloud storage services, general enough to use with eventually consistent key-value stores rather than only SQL storage. While the tool does not itself provide a means to evaluate traffic patterns discussed in section 2.4, it is possible to coordinate parallel workload generators [1] to achieve this effect. We will discuss this more in 6.2.3.

However, it remains unclear overall how to define or replay dynamic workload patterns or traces against an application, rather than performing constant throughput latency measurements, or saturation testing. Two load generation tools have been found to support mixed workloads with probabilistic client activity, and dynamic load generation: Faban[2] (used by Cloudstone), and Tsung[3].

Tsung is a distributed load generator which can be extended with drivers to measure arbitrary network services. Client sessions are highly configurable, for example the client's course of action can be probabilistically determined, and arbitrary think-times included in their actions, simulating real user behaviour. Moreover, it offers a convenient format for specifying arrival rate phases, which is ideal for our measurement of a storage system's elastic response to changes

[1] `https://github.com/brianfrankcooper/YCSB/wiki/Running-a-Workload-in-Parallel` accessed June 2012

[2] `http://www.opensparc.net/sunsource/faban/www/1.0/docs/howdoi/loadvariation.html` accessed June 2012

[3] `http://tsung.erlang-projects.org/user_manual.html` 6.4 "Defining the load progression" accessed June 2012

in traffic. However, there is no bundled driver for Voldemort, and Tsung drivers are written in Erlang, which makes integration with Voldemort's Java client more complex. Furthermore, its measurements may be unsuitable for the desired analysis: Tsung obtains cumulative means and maximal values with 10 second measurement intervals[4], limiting the statistical analysis and graphs which can be produced.

While workload generators supporting configurable workload phases such as Faban, Tsung, and jMeter are appealing, integrating any measurement tool entails an engineering overhead. In this work we adopt YCSB, due to its existing integration with the target system, Voldemort, and prior work by our group with this configuration. Given our measurement approach, described in 6.2.2, its lack of workload phases turns out to be unimportant. However, in the interest of avoiding measurement and instrumentation bias, we delegate the utilisation of alternative measurement software as future work.

6.2.2 Instrumenting Voldemort

Here we describe how the Voldemort storage system was modified to obtain sensor input for our controller, and experimental data for producing graphs and the like.

Our approach is a variation of ongoing work in the group by Ahmad Al-Shishtawy, where a central controller pulling data from measurement clients embedded in the storage client library. Instrumenting the client is reasonable, as we are considering storage for a 3-tier web application, so our clients are the second tier, the application layer's client library, rather than a human consumer outside of the data-centre. The client library is already "smart", as it knows how to route requests for a given key to a storage server holding that data. In the case that dumb clients are used, a front-end server can adopt this role in much the same way that it adopts the responsibility of routing.

More concretely, our measurement clients consist of a thread performing a receive-reply loop to respond to the controller's pull requests, threadsafe data structures for recording workload information, and instrumentation of the client software to record the required information.

The controller periodically polls the measurement clients for measurements, sending each, in parallel, a message requesting their current data. Upon receiving this request, the measurement client replies with their current data and resets their measurement data, ready for the new time-window. In our implementation the measurement data is serialised to a platform independent UTF-8 encoded JSON string, though if found to be a bottleneck this could be optimised to a byte-protocol or other serialisation format.

Having received a set of measurement samples, the controller merges them into a single result: its current view of the system. This view is then used by the

[4] `http://tsung.erlang-projects.org/user_manual.html` 7.3 accessed June 2012

controller as its sensor input, and logged to disk for experimental analysis.

6.2.3 Measuring the system

Having selected a measurement framework (6.2.1) and instrumented the storage system (6.2.2), we consider the specifics of generating a characteristic workload on the system.

Firstly, and once only for the target data store, we pre-populate the store with data. Having done so, we disable the "warmup" code-path, allowing us to pass the number of records to the Voldemort YCSB performance tool's key generators, ensuring requests are on populated keys, without repeating data insertion or even growing the data-set size beyond the amount which fits entirely in memory, complicating matters by introducing multiple storage tiers.

Two load generating nodes were executed against the storage cluster in a 4-server configuration, to find the saturation point of the individual load generators for all get-request workloads. Having identified a maximum number of operations per second, in our case 9k, a lower number is taken as a reliable maximum throughput per load generator, we chose 8k.

For a desired throughput, such as 30k operations per second, we determine how many load generators are required such that no generator exceeds this maximum of 8k operations per second. The load generators are then launched in parallel with an even split of the desired overall throughput, and results are collected from the controller after execution completes. Early samples are discarded to allay BDB cache-warming and the unsynchronised start-up of load generation. It should be noted that because measurement is taken at the controller, we do not need to time-synchronise measurements received from the load generating nodes. While there may be an error involved with time-window alignment in the different samples collected by the controller, we assume this to be negligible.

Having collected measurement logs from the controller, they may be analysed as desired, in our case by scripted format conversion and graph plotting with Ruby and gnuplot.

6.2.4 Evaluation system hardware

Storage nodes were hosted on a 5-node cluster, each node having 4x3.4GHz Intel Xeon, 8GB RAM, Debian Linux 2.6.32-5-686-bigmem, 100Mbps Ethernet, and being located on the same network switch. A 6th node in the cluster has the same configuration, but 2 rather than 4 cores, and was used to host the controller.

Load generators were provisioned from campus workstations (outside of term-time, so unused), Intel(R) Core(TM)2 Quad CPU Q9400 @ 2.66GHz with Gigabit Ethernet.

Future work includes repeating these experiments on a cloud platform, to consider higher scalability, and real cloud actuation delays.

6.2.5 Alternate controller: uniform load

For this experiment our controller operates with a modified executor algorithm, rather than optimising the layout of partitions by their workload, it will introduce new nodes to the cluster and allocate them arbitrary partitions from nodes which have numerically many partitions.

The key notion here is the assumption that each partition will be equally busy, so load-balancing may be achieved by ensuring that each storage node holds a roughly equal number of partitions. This is in contrast to the proposed controller, which may opt to position a very popular partition on a server by itself, and position all other partitions on a single server if they are infrequently accessed.

6.3 Profiling a single Voldemort node

In order to make control decisions based on the number of requests arriving at a node, or whether any given node can meet the demand for a particular partition, we must obtain a workload model for our deployed system.

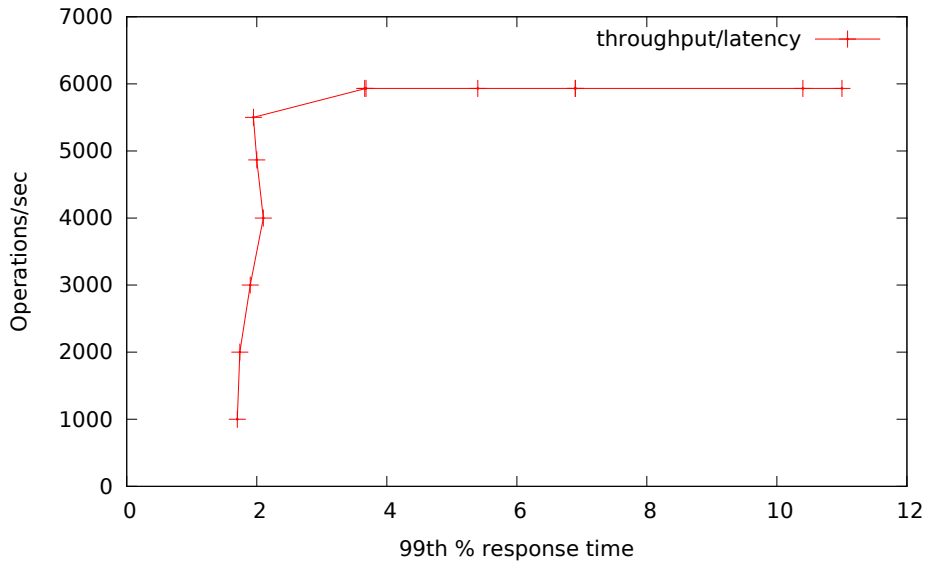

Figure 6.1: Throughput-latency characteristic for one of our Voldemort server nodes; uniformly keyed get requests

Figure 6.1 shows that a node's response time increases with throughput up to a point, then its throughput remains fixed and response times increase,

as the system is saturated and queuing dominates completion time. Having reached a similar throughput figure when generating load with this hardware configuration, it is believed that the 100Mbit NIC is the bottleneck. However, we are not particularly interested in where the bottleneck occurs, only that it does, thus providing the degrading performance we need to exercise our controller.

6.3.1 Instrumentation limitation

Figure 6.1 also demonstrates a limitation of our current instrumentation: operations per second count completions rather than arrival rate, meaning that queuing requests are not counted, and the characteristic produced indicates that performance degrades at about 6k completions per second, but does not reveal how arrival rate and response time are related. Unfortunately, time was not available to remedy this shortcoming, and when considering the following results it should be kept in mind that the when response times increase, the arrival rate exceeds the throughput, so the stacked histograms indicate the completed rather than actual load the system is experiencing.

6.4 Results

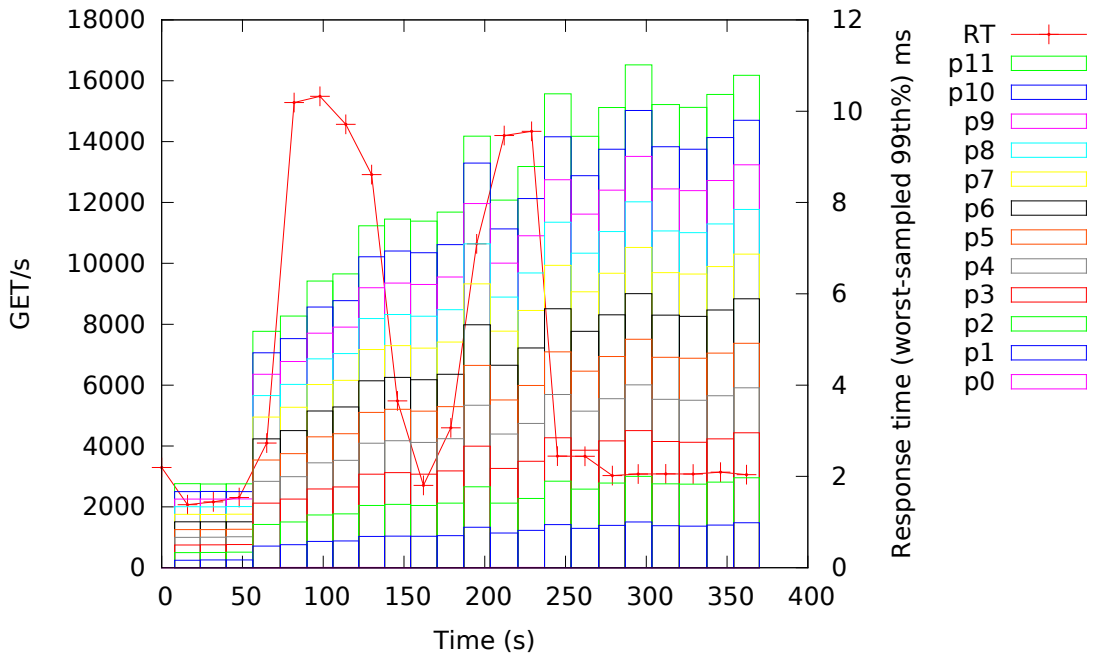

Figure 6.2: Uniformly-distributed diurnal workload increase

To demonstrate that our controller functions sensibly under expected operating conditions, we present in Figure 6.2 its control of a 2-node starting cluster scaling up to meet increasing (uniformly distributed) demand fashioned after the diurnal traffic pattern (2.4.1).

Our graphs present a time progression with two valuable insights into the system behaviour: its response time, and the amount of work it is completing per second (completion rate, or workload – see 6.3.1). The 99th percentile is a good response time metric, as presented by Amazon's Dynamo [11], and focussed on by the SCADS Director (3.2). However, it can be difficult to measure in a distributed setting. Each of our measurement agents records its own 99th percentile response time for each time window, but having collected them we need to take an aggregate value; how to do this is not mathematically obvious, and for expedience we opted to take the maximum value (worst-case) for our graphs.

Workload is displayed as a stacked-histogram, where each block's height represents the number of requests made to a single partition. The height of each stack shows the system's total throughput for the given time window. In a uniform workload all partition blocks the same height, as illustrated by Figure 1.1.

Having established its basic functionality, we move on to the more interesting question of whether it exhibits any measurable benefit over a uniform-load assuming controller (which we consider analogous to "better than random"). To demonstrate the significance of partition loading we present the response of both controllers in two scenarios: their response to a large step-increase in demand for a single key, and their response to a shift from uniform to skewed (mostly single-key) workload where the total workload remains constant.

Both controllers will realise that there is a problem with the system's performance, the difference is in how they react. The "uniform-load" controller will introduce an additional storage node, and steal partitions from other nodes, maintaining the approximate invariant that all storage nodes hold the same number of partitions.

6.4.1 Single-key step increase response

Here the system is put under a steady uniformly-distributed load, and after some time this is suddenly increased by 8k target throughput, by introducing two additional load generators, each aiming for a throughput of 4k.

Figure 6.4 shows the response of the controller making use of partition load information to rebalance, while Figure 6.3 is the response of the controller performing uniform-load assuming partition stealing to rebalance the overloaded cluster.

We see that both controllers recover to reasonable response times, with no significant difference between them. However, two other important differences indicate that the partition-load aware controller performs better: it takes half

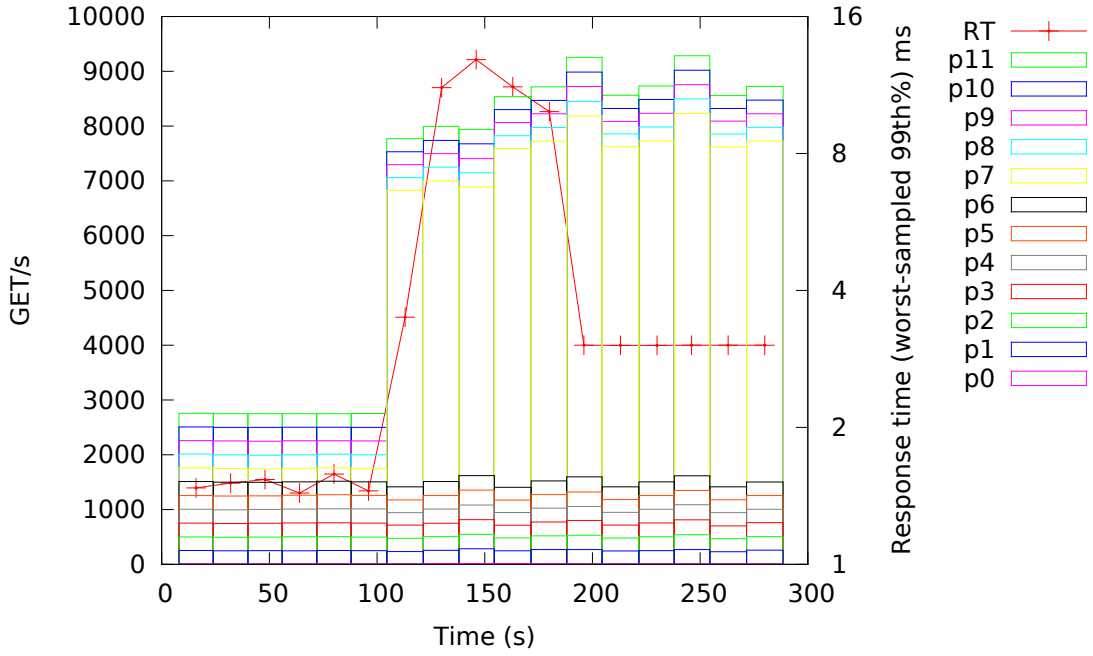

Figure 6.3: Uniform-workload controller reacting to 12k step increase in target throughput to a single key

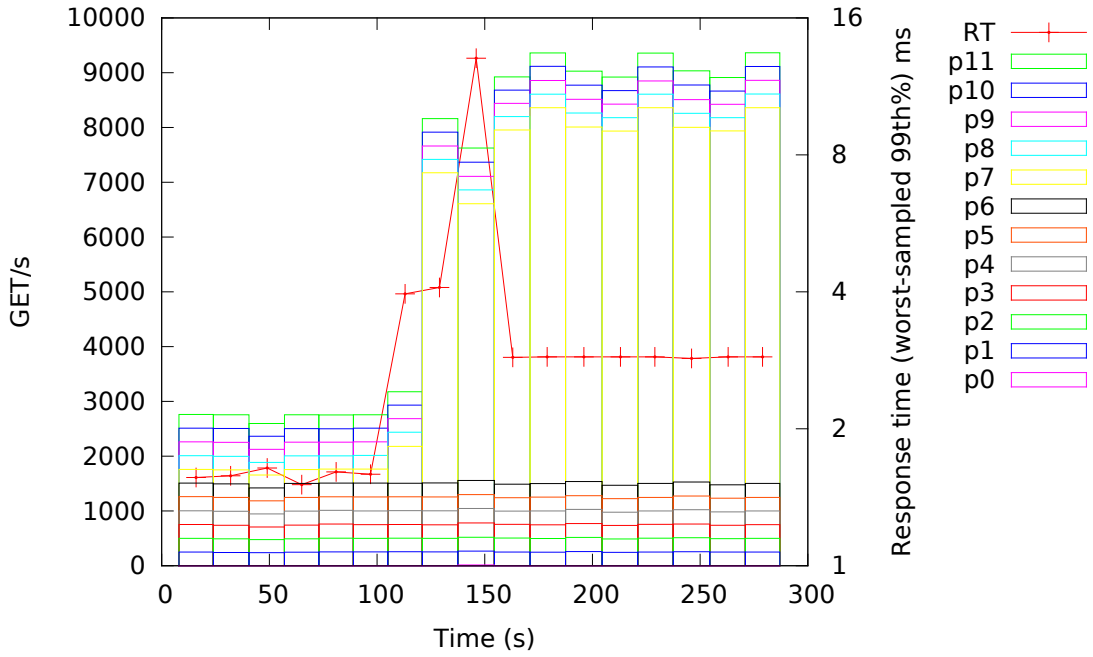

Figure 6.4: Partition-load aware controller reacting to 12k step increase in target throughput to a single key

the time to stabilise, and (not shown in the graph) it uses 3 rather than 5 nodes to serve the increased load.

It is inevitable that both controllers exhibit a worsening of response time after stabilising, as the hot partition in the new workload is too busy to be supported by a single server, which is the best we can offer it with this system's available actuators – increasing its replication, or splitting the partition, would be preferable.

This result lends credence to the claim that storage controllers using fine-grained workload information, such as partition request counts, are able to rebalance more quickly, with less service disturbance, by moving only the data necessary to stabilise performance, resulting in fewer service level violations. Furthermore, it demonstrates that by adopting a workload based bin-packing approach we may provide good response times with fewer machines rented from the cloud provider, saving money.

6.4.2 Constant overall workload with shift from uniform to skewed key access

Figures 6.5 and 6.6 show the controllers' responses to a constant target workload, where the distribution of key accesses shifts from uniform to skewed towards a single key after 3 minutes. The response times achieved by both controllers degrade, though the partition-aware controller again beats the uniform-load assuming controller, this time by half a second. Moreover, its completion rate is higher. While arguably insignificant in these results, these differences may become important at greater scale.

The uniform-load controller is operating on 4 nodes before the access pattern shift occurs, hence its faster stabilisation than in Figure 6.3. Indeed, the uniform-load controller makes use of all 5, and does not realise that it cannot improve performance by adding more, so without this upper bound would continue to over-provision in the attempt to reduce the imbalance caused by the one overloaded partition. In contrast, the partition-load aware controller uses a total of 4 storage nodes, and stabilises upon realising it cannot improve the situation of any of the overloaded nodes due to the workloads of the partitions they hold.

To conclude, we have seen that the partition-load aware controller again achieved marginally superior 99th percentile response times, made use of fewer storage nodes, and detected that it could make no further improvements to the situation, thus stabilising. The uniform-load controller stabilised only because it ran out of available servers, having utilised all 5 of them.

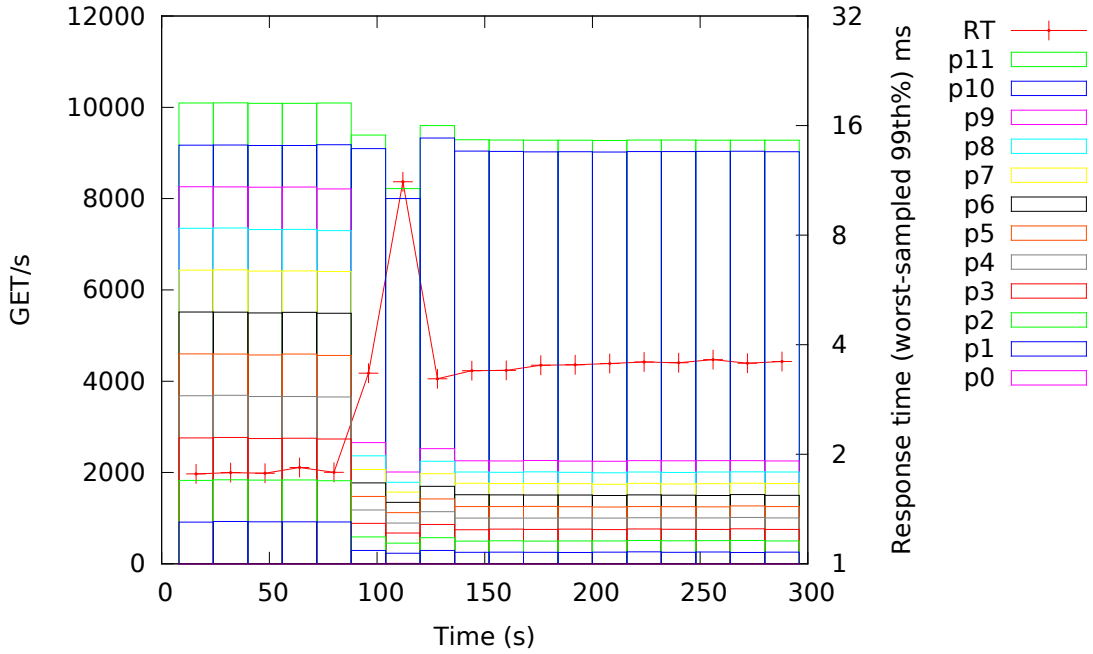

Figure 6.5: Uniform-workload controller response to key access shift, 5 storage nodes used

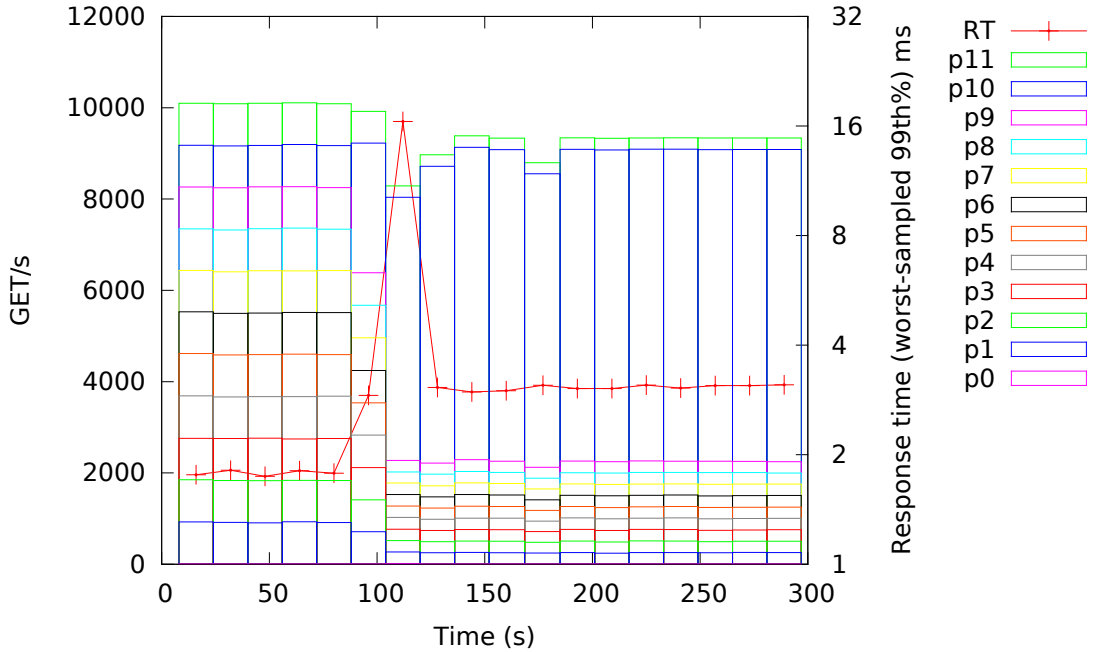

Figure 6.6: Partition-load aware controller response to key access shift, 4 storage nodes used

6.5 Further work

The most immediately obvious limitation of this evaluation is that it was not performed on a cloud platform – servers were statically provisioned and always running, but idle when not part of the cluster. As such, there were no spin up or down delays to their addition to meet demand. Arguably this is a matter to be dealt with by another control component, which would use this component to advise on where and when to move data optimally. Furthermore, rented infrastructure is equivalent to its owned counterpart, but still it seems more appropriate to evaluate a cloud storage controller in the cloud.

That matter covered, other avenues of investigation presented themselves, but were not be pursued due to time constraints.

6.5.1 Reproduction of results with alternative load generator

A shortcoming of these results is the treatment of queueing. Throughput is not clearly differentiated or defined, with the taken completion rate measurements not being representative of the more interesting arrival rate figures. Moreover, the measurement software appears to limit its generated client pool based on the observed throughput, resulting in difficulties in representing the correlation between partition load and performance degradation. As discussed in 6.2.1 other softwares are available, and Tsung in particular is known to produce measurements with better control and measurement of queueing concepts such as arrival rate, completion, and failed requests.

6.5.2 Scaling down

The current controller implementations are incomplete in that they scale out, but not in again. While our experiments have considered pathological situations where this did not matter, other interesting scenarios and more general experimentation require that the controller be able to redress load when empty servers are not available, and also empty out and decommission nodes when workload drops.

6.5.3 Voldemort rebalancer tuning

A possible application of this work which has not been investigated is using our partition workload information to optimise of Voldemort's rebalancing mechanism, either by changing its internal decision-making process, or by batching independent transfer operations together to achieve transmission parallelism and latency-hiding.

6.5.4 Non-uniform file size

In the presented results, a fixed file-size of 1KB is taken for all keys. This may be true of some applications, but does not seem reasonable as a general assumption, in particular since network interfaces were found to be the saturating system component for storage nodes. It would be of interest to investigate the controller's performance with larger fixed file sizes, and with varied file-sizes. This was discussed previously in 5.2.

6.5.5 Fluctuating access patterns

While we have moved beyond uniformly distributed key-access in our experiments, we did not assess the controller's ability to react to and maintain stability in the face of sharp changes or fluctuating workloads, such as traffic surges for a key occurring while the controller is rebalancing to handle a previous, but now moot, shift in key popularity.

This also relates to file-size, or the total stored data size, as when actuation slows down it will become more important to take small steps or be able to interrupt rebalancing in order to revise plans and effect changes appropriate to the new situation.

Another interesting workload would be one consisting of skewed key access, but where the skew towards particular keys changes frequently. From a lesser resolution this would constitute uniform random key access, so such a controller would remain stable and perform reasonably under such load. Our current controller would prove unstable, and worsen rather than improve performance, due to the high observed cost of rebalancing, which would happen each time key popularity changed.

6.5.6 Integration with elastic node provisioning controller

Here a fixed pool of 5 servers were available, acting as an upper bound on the resources available. This limited both the range of loads the servers could be subjected to, and the number of nodes the uniform-load controller could overprovision. Operating this controller in tandem with a controller manipulating the cluster size, such as the Duke HDFS controller of Section 3.1, would lead to new and interesting evaluation scenarios, and a more complete system.

Chapter 7

Discussion and further work

In this work we have investigated the thesis that assuming uniformly-distributed key access is harmful to cloud storage control.

Our results demonstrate that skewed workloads, particularly those for an individual key, as has been the case for stored media relating to popular news stories, may be better managed by automatic controllers taking into account the dissimilar spread of demand for their data partitions.

We have presented contrasting approaches to make decisions with this information: one of fast-to-execute greedy-heuristic decisions, and one of a global optimisation process using a constraint solving system to solve this bin-packing problem. Furthermore, we have presented a simple method for collecting partition-workload information, and provided experimental results for controlling the Voldemort key-value storage system with the greedy-heuristic approach.

7.1 Future directions

In addition to the immediate practical extensions to our results discussed in section 6.5, we here suggest less system-specific directions for future work, or more general experimental situations with another storage system.

7.1.1 Strongly-consistent storage optimisation

While the presented evaluation is for a store without replication, the notion of improving performance without increasing replication seems to be a good fit for strongly consistent storage systems. It would be interesting to integrate the controller with such a store, though it would be necessary to have full control over the system's replica positions – this issue prevented such experimentation with Voldemort.

7.1.2 Replication-degree control for non-uniform loads

With information about the distribution of load across objects or partitions, we can increase or decrease their replication factor. That is, when we notice a partition is busy, we can tell the storage system to increase its replication, or when partitions see little access we may reduce their replication, if some benefit such as storage capacity or fewest distinct nodes were to improve.

However, for this actuation to be effective, the storage system would need to effectively rebalance the replica positions across the storage nodes. This would work best if it knew not to place hot replicas on the same server such that their utilisation exceeds their capacity. Once we assume this behaviour, we may ask why the storage system would monitor this information, unless it already re-partitions its data based on demand.

Another possible situation is less desirable: that replicas will be placed without considering their utilisation, and load will not be balanced across the cluster. Here however the storage service's query semantics are important. Round-robin load balancing will result in the described imbalance, but if nodes serve requests by taking them from a message queue, or similar producer-consumer model, then load balancing may be achieved through the query mechanism, assuming enough replicas of hot partitions, and storage node resources have been provisioned.

7.1.3 Partition resizing and layout optimisation for non-uniform loads

This extends from the partition arrangement approaches of sections 5.3.1 and 5.3.1 by resizing partitions in order to isolate hot items and coalesce cold items, further reducing costly network transfers and easing the burden of layout optimisation for many items.

It is believed that this could be implemented for measurement on top of the SCADS Director or HDFS, but this is left for future work. Voldemort currently blocks this level of control without significant middleware engineering, as storage partitions are fixed.

7.1.4 Controller SLA modelling

Having observed that any low-expenditure oriented storage controller is likely to violate its service levels to some extent during certain traffic scenarios, it would be useful to see a study of framework developed for characterising controllers responses to determine suitable SLAs for usual and unusual traffic, so that controllers reacting as reasonably as can be expected are not excessively penalised in exceptional circumstances.

7.1.5 Global optimisation of partition layout and distance from current layout

Our controller has adopted the greedy local-search heuristic approach to the partition workload bin-packing problem, as in the SCADS Director 3.2. A contrasting approach is to find a globally optimal configuration of storage partitions, ensuring the fewest possible servers are utilised. However, the current location of items and the desire to minimise transferred data places additional constraints on the bin-packing problem, requiring novel model extensions to presenters such as that provided as a case-study for the Gecode constraint programming system in [39].

Given the computational complexity and cost of global optimisation, even with state-of-the-art constraint-programming techniques, we suggest such a mechanism be placed in the planning or deliberation layer, and not block the regular less-optimal controller execution. It might be considered analogous to memory or object garbage collection, or hard disk defragmentation, which are known to be beneficial or required, but are expensive operations which we tend to avoid or schedule to not block real work. This might prove useful, for example in determining which servers can be removed at day and night, preceding or following the diurnal phase change.

7.2 Conclusion

Our results have shown that partition-load aware control offers benefits over uniform-load assuming control in certain situations. We believe the uniform-load assumption to be a very real problem in systems design and evaluation, and have provided initial experimental results supporting this. We have applied concepts from state-of-the-art research to address workloads with non-uniformly distributed key access, enabling us to improve our system's performance recovery under step changes in workload, and to save money by requiring fewer storage nodes to service a given non-uniform workload by bin-packing serviceable demand.

Practically, we see integration with other controllers through fuzzy logic or other multi-agent mechanisms may enable greater cost-savings in cloud storage systems while preserving performance, maintaining service levels. Furthermore, our work as offering a model for how a storage system may allocate added resources, when they are presented by a resource controller interested only in provisioning discrete machines in response to demand, rather than the particulars of how the system uses them.

While we have not demonstrated that the approach is generally superior, or generally applicable, we believe our results demonstrate that a there is a significant difference between the approaches, and the direction is worthy of further investigation.

Bibliography

[1] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson, "Statistical machine learning makes automatic control practical for internet datacenters," ser. HotCloud'09. Berkeley, CA, USA: USENIX Association, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855533.1855545

[2] B. Trushkowsky, P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "The SCADS director: scaling a distributed storage system under stringent performance requirements," in *Proceedings of the 9th USENIX conference on File and stroage technologies*, ser. FAST'11. Berkeley, CA, USA: USENIX Association, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1960475.1960487

[3] M. Moulavi and A. Al-Shishtawy, "State-Space Feedback Control for Elastic Distributed Storage in a Cloud Environment," *ICAS 2012, The Eighth*, no. c, pp. 18–27, 2012. [Online]. Available: http://www.thinkmind.org/index.php?view=article&articleid=icas_2012_1_40_20127

[4] V. Vlassov, A. Al-Shishtawy, P. Brand, and N. Parlavantzas, *Formal and Practical Aspects of Autonomic Computing and Networking*, P. Cong-Vinh, Ed. IGI Global, Oct. 2011. [Online]. Available: http://www.igi-global.com/chapter/niche-platform-self-managing-distributed/60451/

[5] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," 2010, pp. 143–154.

[6] E. Redmond and J. R. Wilson, *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. The Pragmatic Programmers, LLC, 2012. [Online]. Available: http://pragprog.com/book/rwdata/seven-databases-in-seven-weeks

[7] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated control in cloud computing," in *Proceedings of the 1st workshop on Automated control for datacenters and clouds - ACDC '09*. New York, USA: ACM Press, Jun. 2009, p. 13. [Online]. Available: http://dl.acm.org/citation.cfm?id=1555271.1555275

[8] D. Menasce, "QoS issues in Web services," *Internet Computing, IEEE*, vol. 6, no. 6, pp. 72 – 75, Dec. 2002.

[9] N. Bhatti, A. Bouch, and A. Kuchinsky, "Integrating user-perceived quality into web server design," *Computer Networks*, vol. 33, no. 2000, pp. 1–16, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128600000876

[10] A. Van Moorsel, "Metrics for the internet age: Quality of experience and quality of business," in *Fifth International Workshop on Performability Modeling of Computer and Communication Systems, Arbeitsberichte des Instituts für Informatik, Universität Erlangen-Nürnberg, Germany*, vol. 34, no. 13. Citeseer, 2001, pp. 26–31. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.2945&rep=rep1&type=pdf

[11] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 205–220, 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1294281

[12] E. Schurman and J. Brutlag, "The user and business impact of server delays, additional bytes, and HTTP chunking in web search," *Velocity: Web Performance and Operations Conference*, 2009.

[13] F. Duarte, B. Mattos, A. Bestavros, V. Almeida, and J. Almeida, "Traffic Characteristics and Communication Patterns in Blogosphere," Dec. 2006. [Online]. Available: http://dcommon.bu.edu/xmlui/handle/2144/1893

[14] E. Veloso, V. Almeida, W. Meira, A. Bestavros, and S. Jin, "A hierarchical characterization of a live streaming media workload," *Proceedings of the second ACM SIGCOMM Workshop on Internet measurment workshop - IMW '02*, p. 117, 2002. [Online]. Available: http://portal.acm.org/citation.cfm?doid=637201.637220

[15] Z. Liu, B. Huffaker, M. Fomenkov, and N. Brownlee, "Two days in the life of the dns anycast root servers," *Passive and Active*, 2007. [Online]. Available: http://www.springerlink.com/index/E8714RU456841886.pdf

[16] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing - STOC '97*. New York, New York, USA: ACM Press, May 1997, pp. 654–663. [Online]. Available: http://dl.acm.org/citation.cfm?id=258533.258660

[17] A. D. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, 2008.

[18] G. F. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: concepts and design*. Addison-Wesley Longman, 2005.

[19] A. S. Tanenbaum and M. Van Steen, *Distributed Systems Principles and Paradigms*. Prentice Hall, 2009.

[20] L. Lamport, "Paxos made simple," *ACM SIGACT News*, vol. 32, no. 4, pp. 18–25, 2001.

[21] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Computing Surveys*, vol. 37, no. 1, pp. 42–81, Mar. 2005. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1057977.1057980

[22] C. Lu, G. Alvarez, and J. Wilkes, "Aqueduct: online data migration with performance guarantees," ser. FAST '02. Berkeley, CA, USA: USENIX Association, 2002, pp. 219–230. [Online]. Available: http://dl.acm.org/citation.cfm?id=1083323.1083351

[23] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proceeding of the 7th international conference on Autonomic computing - ICAC '10*. New York, USA: ACM Press, Jun. 2010, pp. 1–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1809049.1809051

[24] M. Armbrust, A. Fox, D. Patterson, N. Lanham, H. Oh, B. Trushkowsky, and J. Trutna, "Scads: Scale-independent storage for social computing applications," in *Conference on Innovative Data Systems Research (CIDR)*, 2009. [Online]. Available: http://arxiv.org/abs/0909.1775http://www-db.cs.wisc.edu/cidr/cidr2009/Paper_86.pdf

[25] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. I. Jordan, and D. A. Patterson, "Automatic exploration of datacenter performance regimes," ser. ACDC '09. New York, NY, USA: ACM, 2009, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/1555271.1555273

[26] D. Vengerov, "A reinforcement learning framework for online data migration in hierarchical storage systems," *J. Supercomput.*, vol. 43, no. 1, pp. 1–19, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1007/s11227-007-0135-3

[27] K. Aberer, A. Datta, and M. Hauswirth, "The quest for balancing peer load in structured peer-to-peer systems," Technical Report IC/2003/32 2003, Tech. Rep., 2003.

[28] J. Byers, J. Considine, and M. Mitzenmacher, "Simple load balancing for distributed hash tables," *Peer-to-Peer Systems II*, pp. 80–87, 2003.

[29] D. Karger and M. Ruhl, "New Algorithms for Load Balancing in Peer-to-Peer Systems," MIT, Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, Tech. Rep., Jul. 2003. [Online]. Available: http://hdl.handle.net/1721.1/29831http://dspace.mit.edu/handle/1721.1/29831

[30] J. Kramer and J. Magee, "Self-Managed Systems: an Architectural Challenge," in *Future of Software Engineering (FOSE '07)*. IEEE, May 2007, pp. 259–268. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4221625http://dl.acm.org/citation.cfm?id=1253532.1254723

[31] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White, "A Multi-Agent Systems Approach to Autonomic Computing," ser. AAMAS '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 464–471. [Online]. Available: http://dx.doi.org/10.1109/AAMAS.2004.23http://dl.acm.org/ft_gateway.cfm?id=1018780&type=pdf

[32] A. Al-Shishtawy, V. Vlassov, P. Brand, and S. Haridi, "A Design Methodology for Self-Management in Distributed Environments," *2009 International Conference on Computational Science and Engineering*, pp. 430–436, 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5283371

[33] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. Hoboken, NJ, USA: John Wiley & Sons, Inc., Aug. 2004. [Online]. Available: http://doi.wiley.com/10.1002/047166880X

[34] J. R. Taylor, *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements*, ser. Physics - chemistry - engineering. University Science Books, 1997.

[35] J. Boulon, A. Konwinski, R. Qi, A. Rabkin, E. Yang, and M. Yang, "Chukwa, a large-scale monitoring system," in *Cloud Computing and its Applications (CCA'08)*, vol. 8, 2008.

[36] J. Kephart and W. Walsh, "An artificial intelligence perspective on autonomic computing policies," in *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, june 2004, pp. 3 – 12.

[37] E. Gat, "On three-layer architectures," in *Artificial intelligence and mobile robots*. Cambridge, MA: AIII Press, 1998, ch. 8. [Online]. Available: http://www.tu-chemnitz.de/etit/proaut/paperdb/download/gat98.pdf

[38] LinkedIn, "Project Voldemort." [Online]. Available: http://project-voldemort.com/

[39] C. Schulte, G. Tack, and M. Z. Lagerkvist, "Case studies," in *Modeling and Programming with Gecode*, C. Schulte, G. Tack, and M. Z. Lagerkvist, Eds., 2012.

[40] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, S. Patil, A. Fox, and D. Patterson, "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," 2008.