Informed data distribution selection in a self-predicting storage system

Eno Thereska[1], Michael Abd-El-Malek[1], Jay J. Wylie[2],
Dushyanth Narayanan[3], Gregory R. Ganger[1]

[1] Carnegie Mellon University, [2] HP Labs - Palo Alto, [3] Microsoft Research - Cambridge

CMU-PDL-06-101

January 2006

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

Systems should be self-predicting. They should continuously monitor themselves and provide quantitative answers to What...if questions about hypothetical workload or resource changes. Self-prediction would significantly simplify administrators' planning challenges, such as performance tuning and acquisition decisions, by reducing the detailed workload and internal system knowledge required. This paper describes and evaluates support for self-prediction in a cluster-based storage system and its application to What...if questions about data distribution selection.

Acknowledgements: We thank the members and companies of the PDL Consortium (including APC, EMC, Engenio, Equallogic, Hewlett-Packard, HGST, Hitachi, IBM, Intel, Microsoft, Network Appliance, Oracle, Panasas, Seagate, Sun, and Veritas) for their interest, insights, feedback, and support. We also thank Intel, IBM, and Seagate for hardware donations that enabled this work.

Keywords: data distribution, end-to-end tracing, self-prediction

1 Introduction

Storage administration is a difficult and expensive task [Ganger03a, Gartner00, Gray03]. One major portion of this task addresses planning challenges such as acquisition decisions, component configurations, assignment of datasets/workloads to components, and resolving performance problems. For many of these challenges, the most complex aspect is understanding the performance consequences of any given decision.[1] These consequences usually depend on specifics of the workload (i.e., the interleaved I/O patterns of the various applications) and the storage system internals.

Traditionally, administrators use two tools when making planning decisions: their expertise and system over-provisioning. Most administrators work with a collection of rules-of-thumb learned and developed over their years of experience. Combined with whatever understanding of application and storage system specifics are available to them, they apply these rules-of-thumb to planning challenges. For example, one administrator might apply the rule "if average queueing delays are greater than 10 ms, then spread data/work over more disks" to resolve a perceived performance problem. Since human-utilized rules-of-thumb are rarely precise, over-provisioning is used to reduce the need for detailed decisions. For example, one common historical rule-of-thumb called for ensuring that disk utilization stayed below 30% (i.e., always have three times the necessary disk throughput available). Both tools are expensive, expertise because it requires specialization and over-provisioning because it wastes hardware and human[2] resources. Further, sufficient expertise becomes increasingly difficult to achieve as storage systems and applications grow in complexity.

We believe that systems must provide more assistance to administrators. Storage systems should be *self-predicting*: able to provide quantitative answers to administrators' performance questions involved with their planning. With appropriate built-in monitoring and modeling tools, we believe that storage systems can answer *What*...*if* questions about potential changes. For example, "*What* would be the performance of workload X *if* its data were moved from device A to device B?". With answers to such *What*...*if* questions, administrators could make informed decisions with much less expertise. Further, iterating over *What*...*if* questions (e.g., one for each possible option) enables a search-based approach to automating, or at least automatically guiding, planning decisions.

This paper describes support for self-prediction in a cluster-based storage system and its application to *What*...*if* questions about data distribution choices. The *data distribution* for a dataset describes how it is encoded (e.g., replication vs. erasure coding) and assigned to storage-nodes within the cluster. No single data distribution choice is best for all data [Abd-el-malek05b], and cluster-based storage systems will support a variety of choices just like disk array systems (RAID 5, RAID 0+1, etc.). The data distribution used for a given dataset has a large impact on its performance, availability, and confidentiality. Self-prediction assists with understanding the performance impacts of any given data distribution option.

Of course, the performance for a data distribution is a complex function of I/O workload and storage-node characteristics. Selecting the right encoding requires knowledge of the access patterns and the bottleneck resources. For example, small random writes often interact poorly with erasure coding, but large streaming writes benefit from the reduced network bandwidth used relative to replication. Data placement requires the same knowledge as encoding selection, as well as knowledge of how workloads will interact when sharing storage-nodes. For example, two workloads that benefit from large caches may experience dramatic performance decreases if assigned to the same storage-node. Answering *What*...*if* questions about data distribution choices requires accounting for all of these effects.

Self-prediction has two primary building blocks: monitoring and modeling. The monitoring must be detailed so that per-workload, per-resource demands and latencies can be quantified. The aggregate performance counters usually exposed by storage systems are insufficient for this purpose. Our system

[1]Non-performance issues, such as cost and reliability, are also involved. But, these usually require much less understanding of the inner workings of system components and applications.

[2]The additional hardware must be configured and maintained.

uses end-to-end instrumentation in the form of traces of "activity records" that mark steps reached in the processing of any given request, and post-processes the traces to compute demands and latencies. Modules for answering *What*...*if* questions use modeling tools and the observation data to produce answers. Tools used include experimental measurements (for encode/decode CPU costs), operational laws (for bottleneck analysis), and simulation (for cache hit rate projections). *What*...*if* questions can be layered, with high-level *What*...*if* modules combining the answers of multiple lower-level *What*...*if* modules. For example, "*What* would be the performance of client A's workload *if* we add client B's workload onto the storage-nodes it is using?" would need answers to questions about how the cache hit rate would change, how the network utilization would change, etc.

Evaluations show that our self-prediction infrastructure is effective. Most importantly, high-level *What*...*if* questions about how performance for different workloads will change with dataset migration or encoding changes are answered with less than 15% error in almost all cases. Such accuracy should be sufficient for most planning decisions, as they exceed the detail usually available for the traditional approach. The monitoring instrumentation places less than 6% overhead on foreground workloads, which we view as an acceptable cost for the benefits provided.

These results demonstrate the feasibility of self-prediction for substantial *What*...*if* questions in real systems. In the specific context of data distribution planning, these *What*...*if* questions could be used in making assignment decisions for new workloads, redistribution decisions when new storage-nodes are acquired or existing storage-nodes fail, migration and re-encode decisions for tuning performance, etc. We believe that such self-prediction will play an important role in the move towards more automated systems.

2 Data distribution selection

It is difficult to understand the performance implications of a data distribution choice. To do so requires a detailed understanding of the interactions between a workload and the system resources, and an understanding of how those interactions change with the encoding choice. Both choosing the right encoding and the right set of storage-nodes on which to actually place the data are dynamic problems. Clients enter and leave a system and storage-nodes are added and retired during failures. Clients' workloads also change and may require re-encoding and re-distribution onto different sets of storage-nodes for load balancing. To improve the process of data distribution selection, we've developed a generic infrastructure that can evaluate the performance impact of hypothetical choices.

2.1. Cluster-based storage systems

Traditional storage systems are built around a single-vendor, monolithic disk array design that has traditionally focused on high-performance and availability. Such systems are very reliable, but they are also expensive and do not scale well. Incremental scaling is not an option with such systems, and a client often needs to buy another expensive system when the scalability requirements slightly exceed the ones provided by the older system. Cluster-based storage systems, built from commodity hardware, have been developed to address these scalability and cost issues [Abd-el-malek05b, Saito04a, Zhang04b]. The individual servers are often called *storage-nodes* and provide a certain amount of CPU, buffer cache and storage space. These components are cheap since they can be mass-produced. Incremental scalability is provided by their addition into the system.

Commodity hardware is often less reliable than customized hardware, and these storage-nodes usually have lower performance than customized disk arrays. To make up for the lower storage-node reliability and performance, data is strategically distributed to enable access parallelism and reliability in face of node failures. A data distribution is an algorithm for encoding the data to meet availability and confidentiality

Parameters	**Description**
n	Data is encoded in n fragments.
m	Any m fragments reconstruct the data.
$encryption$	Encryption ensures confidentiality.

Table 1: **Data encoding tunable parameters.**

needs and choosing the set of storage-nodes to host the data.

There is no single data distribution that is best for all data. The data distribution choice has major impact on three crucial system metrics: availability, confidentiality and performance. The data a bank stores, for example, has different availability goals than the data of an online retailer [Keeton04], and thus they may require a different encoding. The online retailer may have a stricter confidentiality goal than an email provider and thus may require encryption. The online retailer may have more stringent performance requirements than the bank, and require that response times be kept below a threshold.

2.2. Data encoding

A data encoding specifies the degree of redundancy with which a piece of data is encoded, the manner in which redundancy is achieved, and whether or not the data is encrypted. Availability requirements dictate the degree of data redundancy. Redundancy is achieved by replicating or erasure coding the data [Chen94, Rabin89]. The erasure coding scheme is characterized by the parameters (m, n). An m-of-n scheme encodes data into *n fragments* such that reading any m of them can reconstruct the original data. Confidentiality requirements dictate whether or not encryption is employed. Encryption, is performed prior to such encoding (and decryption is performed after decoding). Table 1 lists these tunable parameters.

There is a large trade-off space in terms of the level of availability, confidentiality, and system resources (such as CPU, network, storage) consumed as a result of the encoding choice. For example, as n increases, relative to m, data availability increases. However, the storage capacity consumed also increases (as does the network bandwidth required during data writes). As m increases, the encoding becomes more space-efficient: less storage capacity is required to provide a specific degree of data redundancy. However, availability decreases (more fragments are needed to reconstruct the data during a read). When encryption is used, the confidentiality of the data increases, but the demand on CPU increases (to encrypt the data). Other trade-offs with respect to CPU, storage and network demand are discussed in Section 3.3.4 and Section 3.3.5.

The workload for a given piece of data should also be considered when selecting the data encoding. For example, it may make more sense to increase m for a write-mostly workload, so that less network bandwidth is consumed. Compare 3-way replication (i.e., a 1-of-3 encoding) that consumes approximately 40% more network bandwidth to a 3-of-5 erasure coding scheme for an all-write workload. For an all-read workload, however, both schemes consume the same network bandwidth. Others have explained these trade-offs in significant detail [Weatherspoon02, Wylie05].

Because of this large trade-off space and the dependence on workload characteristics, it is very difficult for an administrator to know *a priori* the effects of an encoding change — hence the need for system self-prediction. Ideally, a system would answer high-level performance questions related to throughput and latency by answering sub-questions of the form "*What* would be the CPU/network/storage demand of workload A, *if* it is encoded using scheme E?".

2.3. Data placement

In addition to selecting the data encoding, the storage-nodes on which encoded data fragments are placed must be selected. When data is initially created, the question of placement must be answered. Many different system events may require the placement decision to be revisited. For example, when new storage-nodes are added to the cluster, when old storage-nodes are retired, and when the workloads have changed sufficiently to warrant re-balancing load. Quantifying the performance effect of adding or subtracting a workload from a set of storage-nodes is non-trivial. Each storage-node may have different physical characteristics (e.g., the amount of buffer cache, types of disks, and network connectivity) and host different pieces of data whose workloads lead to different levels of contention for the physical resources.

Workload movement *What*...*if* questions (e.g., "*What* is the expected throughput/response client A can get *if* its workload is moved to a set of storage-nodes *S*?") need answers to several sub-questions. First, the buffer cache hit rate of the new workload and the original workloads on those storage-nodes need to be evaluated (i.e., for each of the workloads the question is "*What* is the buffer cache hit rate *if* I add/subtract workload A to/from this storage-node?"). The answer to this question will depend on the particulars of the buffer cache management algorithm the storage-node uses. Second, the disk demand (or service time) for each of the I/O workloads' requests that miss in buffer cache will need to be predicted (i.e., for each of the workloads, the question is "*What* is the average I/O service time *if* I add/subtract workload A to/from this storage-node?"). Third, the network load on each of the storage-nodes that results from adding/subtracting workloads needs to be predicted as well.

It is challenging for administrators to answer *What*...*if* questions such as the above. Doing so requires one to understand the system internals (e.g., buffer cache replacement policies) and keep track of the workloads each resource is seeing (e.g., buffer cache records for each workload and storage-node). The next section describes how encoding and workload addition/subtraction problems can be answered with end-to-end instrumentation and built-in models.

3 System design and architecture

This section describes a cluster-based storage system and how its design supports performance self-prediction.

3.1. Versatile cluster-based storage

Ursa Minor is a cluster-based storage system that provides data distribution versatility (i.e., a wide range of options within a single system) and the ability to change data to a different distribution online. Its architecture and implementation are described by Abd-El-Malek et al [Abd-el-malek05b]. At the core of its architecture is the separation of mechanical functions (servicing client requests) from managerial functions (automating administrative activities). The managerial tier consists of agents and algorithms for automating internal decisions and helping administrators understand the consequences of external ones. The mechanical tier is designed to self-monitor and includes self-predictive capabilities used by the management tier. The high-level architecture of our system is shown in Figure 1. Below, we explain some of the terminology used.

Clients: Clients of the system store and access data. Data may have different availability, confidentiality and performance goals. Clients make use of the *PASIS protocol family* to encode data [Goodson04, Wylie05]. (We use only a subset of the versatility provided by the PASIS protocol family in this work, i.e., we exclusively consider crash failures in a synchronous timing model.) Illustrated in Figure 1 are two clients. The first is writing data with a 3-of-5 encoding (thus having to write to 5 storage-nodes). The second is reading the data from 3 of the 5 storage-nodes.

Storage-nodes: The storage-nodes have CPUs, buffer cache and disks. Storage-nodes are expected to be heterogeneous, as they get upgraded or retired over time and sometimes are purchased from different vendors.

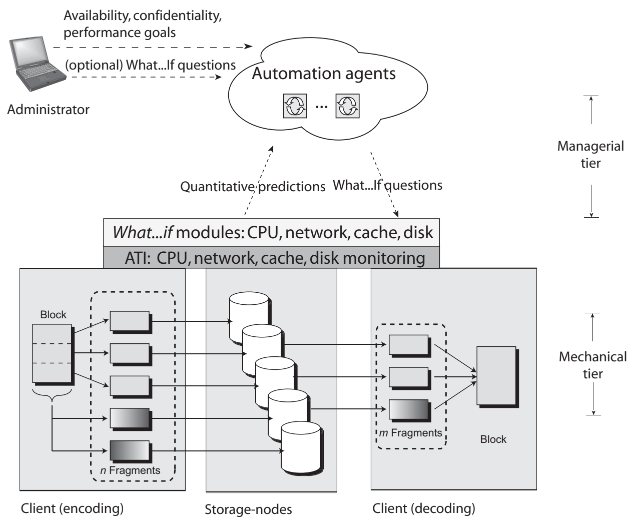

Figure 1: **High-level architecture of the self-predictive architecture within Ursa Minor.** The mechanical tier, on the bottom, services I/O requests for clients. The managerial tier, on the top, provides automation. It makes use of the self-predicting capabilities of the individual system components to get answers to various *What*...*if* explorations.

Administrators: Administrators are responsible for setting availability, confidentiality and performance goals. Availability goals may be expressed with a monetary value attached to data loss or data outage (e.g., as Keeton et al. describe [Keeton04]). Confidentiality may be specified as a binary choice (encrypt the data or not). Performance goals are often expressed in terms of service-level agreements that specify a desired level of throughput and response time. Administrators are not required to understand the workload-system interactions. They can use the predictive infrastructure to ask *What*...*if* questions (e.g., for guiding purchase decisions).

Automation agents: Automation agents are responsible for making sure that administrator goals are met. Previous work has shown how to convert availability and confidentiality goals into encoding decisions [Goodson04, Keeton04, Wylie05]. In this work, we focus on enabling the automation agents to quantify the performance impact of data distribution choices.

Activity tracking infrastructure (ATI) and *What*...*if* modules: The ATI continuously tracks requests as they move from component to component in the distributed storage system. The ATI is integrated in every storage-node. It allows differentiation among multiple workload streams and presents a unified distributed performance monitoring infrastructure. *What*...*if* modules use that infrastructure to measure resource consumption by different clients and make performance predictions regarding hypothetical workload and/or resource changes. Predictions from several resource-specific *What*...*if* modules are analyzed by the automation agents to make high-level throughput and response time predictions.

3.2. Continuous resource monitoring

The ATI is responsible for keeping track of the performance of every client request along the entire execution path of the request. The ATI thus keeps track of per-client per-resource demands, as well as per-client request latency graphs. Such graphs can help understand where a request spends most of its time. The ATI retains activity records, such as buffer cache reference records, I/O records, and network transmit/receive records. The sequence of records allow tracking of a request as it moves in the system, from one computer, through the network, to another computer, and back. Retaining activity records permits automation agents to use

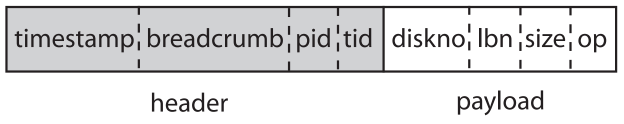

Figure 2: **Example activity record.** Each activity record has a common header and a payload. The payload for the disk request activity record shown includes the disk id, logical block number (lbn), size of the I/O in bytes, and operation type.

simulation techniques to answer *What*...*if* questions, when needed. Activity records are effectively a superset of performance counters. Any performance counter value of interest can be extracted by querying the database of records.

Each computer runs a single ATI instance. The ATI instance is responsible for presenting any process running on that computer with APIs for posting and querying activity records. For querying flexibility, ATI records are stored in relational databases (Activity DBs). Activity records posted to the ATI instance are periodically sent to Activity DBs. Activity DBs run on the same infrastructure computers with the rest of the system. The DBs store the records in relational tables and answer queries on those records.

An *activity record* is the smallest unit of data stored in the Activity DBs. An activity record is a sequence of (attribute, value) pairs. Figure2 shows an example activity record. Each activity record contains an automatically-generated header comprised of a timestamp, breadcrumb, kernel-level process id, and user-level thread id. The timestamp is a cycle-accurate timestamp generated by the computer that permits accurate timing measurements of requests. The breadcrumb permits records associated with a given request to be correlated within and across computers. Activity records are posted at strategic locations in the code so that the demand on a resource is captured. For example, the disk activity record is posted both when the request is sent to disk and when the request completes. Both postings contain the same breadcrumb, because they belong to the same request, and so can be correlated. Posting activity records is on the critical path; however, as our evaluation shows, such posting causes minimal impact on foreground performance. Table2 lists the instrumentation points in Ursa Minor.

Activity DBs are *queried* by internal resource-specific *What*...*if* modules using the common SQL language. For example, to get a disk I/O trace for a certain storage-node, one could query the Activity DB that keeps records for that storage-node's disk activity records. Querying activity records is not on the critical path, though storing activity records in a database permits efficient execution of queries.

3.3. What...if modules

What...*if* modules are structured in two tiers. The lower tier answers performance questions pertaining to specific resources (e.g., CPU, buffer cache hit rate, I/O service times). The upper tier is part of the Automation Agents and is responsible for using the lower tier to make high-level predictions of performance metrics of interest.

3.3.1 Performance metrics

Performance metrics of interest to us are expected client *throughput* and *response time* under a hypothetical data distribution change. In addition, we want to predict client *peak achievable throughput*. In Ursa Minor, there is a request processing pipeline with at least two stages (a request to access data needs to access the metadata service and then the storage-nodes). Throughput depends on the number of outstanding requests the client issues to fill that pipeline. Intuitively, peak throughput is achieved when the pipeline is full. Any further increase in number of outstanding requests does not increase throughput but may increase response time.

	Record Type	Arguments	Description
CPU demand	*UserThreadSwitch*	*oldthread, newthread*	A user-level context switch
	KernelProcessSwitch	*cpuid, oldprocess, newprocess*	A kernel context switch
Buffer cache demand	*BufferReadHit*	*file, offset, size*	Denotes a buffer cache hit
	BufferReadMiss	*file, offset, size*	Denotes a buffer cache miss
	BufferWrite	*file, offset, size*	Denotes a write. Marks buffer dirty
	BufferReadAhead	*file, offset, numblocks*	Prefetch pages (non-blocking)
	BufferEvict	*file, offset, size*	Evict a page to disk
	BufferFree	*file, offset, size*	Release a page to the free pool
Network demand	*NetworkTransmit*	*sender, receiver, numbytes*	Monitors network flow
Disk demand	*DiskOp*	*diskid, lbn, size, operation*	Monitors disk activity

Table 2: **Activity record types posted in Ursa Minor.** KernelProcessSwitch records are provided by the (modified) OS kernel; the other records are posted from instrumentation points in user-level processes. Each record is also automatically annotated with a common header as shown in Figure2. There are approximately 200 instrumentation points in Ursa Minor.

3.3.2 Throughput prediction

To predict aggregate throughput under a hypothetical distribution change, our algorithms assume a closed-loop workload[3] and use operational analysis [Lazowska84a] on all resources (CPUs, networks, disks).

Let D_i^k be the average demand, in seconds, of a request from client i on resource k. Let D_i be the sum of all demands on all resources a request uses. Let D_i^{max} be the largest demand client i places on any resource (that resource with the highest demand is called the bottleneck resource). If the ATI measures that client i has, on average, N_i requests outstanding, then client i's throughput bound T_i is:

$$T_i \quad \leq \quad min\left(\frac{1}{D_i^{max}}, \frac{N_i}{D_i}\right) \tag{1}$$

If the client has a small number of outstanding requests, and thus cannot keep all resources utilized, then its throughput is predicted to be the second part of the equation (N_i/D_i). Otherwise, the throughput is the peak throughput $1/D_i^{max}$ obtained by saturating the bottleneck resource. The threshold N_i^* for determining if the load is light or not is $N_i^* = D_i/D_i^{max}$. N_i^* can be thought of as the minimum number of requests required to keep the request pipeline full.

Let T_i^{CPU} be the maximum throughput of the client CPU, in terms of requests it can process. It equals $1/D_i^{CPU}$, where D_i^{CPU} is the average CPU demand per request. The new CPU demand is predicted using the method described in Section 3.3.4.

Let T_i^{NET} be the maximum network throughput, in terms of number of requests it can process. It equals $1/D_i^{NET}$, where D_i^{NET} is the average network demand per request. The original network demand is measured while the workload has been running. The new network demand is predicted based on the the observed workload patterns and the new encoding decision as described in Section3.3.5.

Let $T_i^{I/O}$ be the maximum disk throughput, in terms of number of requests that it can process. It equals $1/n_i^{I/O} D_i^{I/O}$. $n_i^{I/O}$ is the average number of disk requests that result from the original client request. Not all original client requests result in a disk request. Some requests hit in the buffer cache and do not go to disk. In general, if the buffer cache hit rate for a client i is p_i, then n_i equals $1 - p_i$. $D_i^{I/O}$ is the average service time for a disk request.

[3]If the workload is open-loop, then throughput is the number of requests the client is sending and does not need to be predicted.

Both $n_i^{I/O}$ and $D_i^{I/O}$ need to be predicted for a hypothetical data distribution. Both depend heavily on the interaction between the node's buffer cache size and disks at each node, as explained in Sections3.3.6 and 3.3.7 respectively. They also depend on workload access patterns (e.g., sequential or random).

3.3.3 Response time prediction

We predict response time R_i by transforming our throughput predictions above using Little's law [Lazowska84a], which states that

$$R_i = \frac{N_i}{T_i} \tag{2}$$

Equation 2 determines the minimum response time when the client achieves peak throughput. Any further increases in the number of client outstanding requests will not increase throughput, but will increase response time linearly.

3.3.4 CPU What...if module

The goal of the client CPU module[4] is to answer sub-questions of the form "*What* is the request demand D_i^{CPU} for requests from client i *if* those requests are encoded using scheme E?". The CPU modules use direct measurements of encode/decode costs to answer these questions. Direct measurements of the CPU cost are acceptable, since each encode/decode operation is short in duration. Inputs to the CPU module are the hypothetical encoding E and the measured read:write ratio of the workload (as measured by the ATI). The CPU module encodes and decodes one request several times with the new hypothetical encoding and produces the average CPU time for reads and writes. Intuitively, schemes based on replication utilize little client CPU, but place more demand on the network and storage resources. Schemes based on erasure coding are network and storage efficient, but require more client CPU work to encode the data. All schemes require significant amounts of CPU work when using encryption.

3.3.5 Network What...if module

The goal of the network module is to answer sub-questions of the form "*What* is the request demand D_i^{NET} for requests from client i *if* those requests are encoded using scheme E?". To capture first order effects, the network module uses a simple analytical function to predict network demand based on the number of bytes transmitted. Inputs to the network module are the hypothetical encoding E and the measured read:write ratio of the workload (as measured by the ATI). In Ursa Minor, a write updates n storage-nodes and a read reads data from only m storage-nodes. The network demand for a single request is the minimum time needed to transmit the data for a request (i.e., if that request was the only one using the network) plus a well-known fixed cost for the latency to get the first bit to the destination. The time to transmit data equals the size of the request in bytes divided by the network bandwidth. The fragment's size is computed as the original request's size divided by m.

3.3.6 Buffer Cache What...if module

The goal of the buffer cache module is to answer sub-questions of the form "*What* is the average number of requests $n_i^{I/O}$ that miss in buffer cache (and thus have to go to disk) *if* a workload from client i is added to a storage-node?". The buffer cache module can similarly answer questions on other workloads when one client's workload is removed from a storage-node. A buffer cache miss requires orders of magnitude more

[4]There is CPU consumed at the storage-nodes as well, for example, when checking data checksums. However, the storage-node CPU does not become the bottleneck in practice, so we focus on the client CPU, which is used for encoding/decoding and encryption.

time than a buffer cache hit, hence the performance the client sees is very much dependent on the storage-node's buffer cache. The buffer cache behavior of a workload depends on its access patterns, working set size, and the storage-node's buffer cache replacement policy.

Consider workload W_0 being placed on a storage-node that already hosts P workloads $W_1,...,W_P$. The prediction takes the form:

$$\{n_0^{I/O}, n_1^{I/O}, ..., n_P^{I/O}\} = BufferCache_{module}\{W_0, W_1, ..., W_P\} \quad (3)$$

The buffer cache module uses simulation to make a prediction. The module takes buffer cache records of each of the $W_0, W_1, ..., W_N$ workloads, collected through the ATI and replays them, keeping track of the new number of hits and misses when these records are run on the same nodes. The reason simulation is used, rather than an analytical model, is because buffer cache replacement and persistence policies are often complex and system-dependent. They cannot be accurately captured using analytical formulas. The storage-node buffer cache policy in our system is a variant of least-recently-used (LRU) with certain optimizations.

3.3.7 Disk What...if module

The goal of the disk module is to answer sub-questions of the form "*What* is the average service time $D_i^{I/O}$ of a request from client i *if* that request is part of a random/sequential, read/write stream?" The average service time for a request is dependent on the access patterns of the workload and the policy of the underlying storage-node. Storage-nodes in Ursa Minor are optimized for writes, utilize NVRAM, and use a log-structured layout on disk [Soules03].

The disk module is analytical in nature. It receives the interleaved sequence of I/Os from the different workloads, scans the combined trace to find sequential and random streams within it, and assigns an expected service time to each of them:

$$\{D_0^{I/O}, D_1^{I/O}, ..., D_P^{I/O}\} = Disk_{module}\{n_0^{I/O}, n_1^{I/O}, ..., n_P^{I/O}\} \quad (4)$$

3.3.8 Using the What...if modules together

To predict client A's throughput, the Automation Agent consults the resource-specific *What...if* modules to determine which of the resources will be the bottleneck resource. Client A's peak throughput will be limited by the throughput of that resource. In practice, other clients will be sharing the resources too, effectively reducing the maximum throughput those resources would provide if client A was the only one running. The Automation Agent adjusts the predicted client A's throughput to account for that loss.

4 Evaluation

This section evaluates the predictive framework. First, we show the accuracy of the individual *What...if* modules under several encoding choices. Second, we show the accuracy of high-level *What...if* questions on throughput and response time that make use of several of the above modules at once. Third, we show that the overhead of the requisite instrumentation is low.

4.1. Experimental setup

The experiments use a cluster of standard x86-based computers. Clients have machines with Pentium 4 Xeon 3.0 GHz processors with 2 GB of RAM. Unless otherwise mentioned, all storage-nodes have Pentium 4 2.6 GHz processors with 1 GB of RAM; they have a single Intel 82546 gigabit Ethernet adapter in each computer, connected via a Dell PowerConnect 5224 switch. The disk configuration in each computer varies and disk capacities range from 8 to 250 GB. All computers run the Debian "testing" distribution and use Linux kernel version 2.4.22. We use micro- and macro-benchmarks. Macro benchmarks are unmodified and make use of an NFS server that communicates directly with the storage-nodes using the Ursa Minor access protocol. Micro benchmarks access the storage-nodes directly using the same protocol.

SSIO_BENCHMARK: This micro benchmark allows control of the workload read:write ratio, access patterns and client outstanding requests. The performance of this workload is reported in terms of requests/sec or MB/s and response time per request. The access size is 32 KB for this benchmark.

OLTP workload: The OLTP workload is an on-line TPC-C-like [TPCCdoc] database workload. Transactions consists of 8 KB read-modify-write operations to a small number of records in a 5 GB database. The performance of this workload is reported in transactions per minute (tpm).

Postmark: Postmark is a user-level file system benchmark designed to emulate small file workloads such as e-mail and netnews. It measures the number of transactions per second that the system is capable of supporting [Katcher97]. A transaction is a file creation or deletion, paired with a read or an append. The configuration parameters used were 20000 files, 20000 transactions, and 140 subdirectories. All other parameters were left as default. The performance of this workload is reported in transactions per second (tps).

Iozone: Iozone is a general file system benchmark that can be used to represent streaming data access (e.g., data mining) [Norcott02]. For our experiments, it measured the performance for 64 KB sequential writes and reads to a single 2 GB file. The performance of this workload is reported in megabytes per second read and written.

For conciseness, we present results for only five data encodings. These results are indicative of the many other encodings we explored. 1-of-1 refers to 1-way replication. 1-of-3 is 3-way replication and can tolerate up to 2 storage-node faults. 1-of-3 encr is 3-way replication where the data is also encrypted to ensure confidentiality. 3-of-5 is an example of an erasure coding scheme that also tolerates two storage-node faults but is more storage efficient than 1-of-3. 3-of-5 encr is the 3-of-5 scheme with encryption. Unless otherwise mentioned, all experiments are run ten times and the average together with the standard deviation is reported.

4.2. Resource-specific What. . . if modules

This section illustrates the output of the resource-specific *What...if* modules in isolation. The CPU and network *What...if* modules are based on direct measurements, hence the prediction accuracy is almost perfect, so for those two resources we concentrate more on illustrating the large variance in resource consumption as a function of encoding choice. The memory and disk *What...if* modules are based on simulation and analytical models respectively, and we concentrate on the prediction accuracy of these modules.

CPU *What...if* module: Recall from Section 3.3.4 that the goal of the CPU module is to answer sub-questions of the form "*What* is the request demand D_i^{CPU} for requests from client i *if* those requests are encoded using scheme E?". Figure 3 shows how the CPU demand varies based on the encoding scheme used. The module runs 100 encode/decode samples each time and reports the average. Some encoding schemes differ from others by more than an order of magnitude, and as we show later in this evaluation, the client CPU may become the bottleneck. The main source of error in predicting the CPU demand is the module being preempted while performing the measurements. However, that error is very small.

Network *What...if* module: Recall from section 3.3.5 that the goal of the network module is to answer sub-questions of the form "*What* is the request demand D_i^{NET} for requests from client i *if* those requests are encoded using scheme E?". Figure 4 shows how the network demand varies based on the encoding schemes used. Note that some schemes such as 3-way replication place a large demand on the network during writes because they need to update all storage-nodes involved. As we show later in this evaluation, the network may become the bottleneck in those cases.

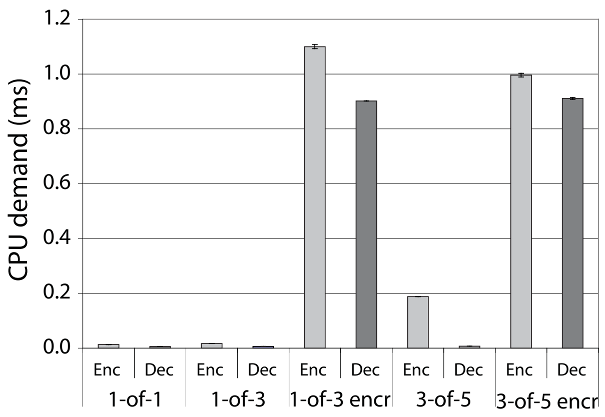

Figure 3: **CPU *What*...*if* module output.** This figure illustrates how the CPU demand per request can differ based on the chosen encoding. Five encoding choices are shown, together with the cost for encoding data (during a write) and decoding it (during a read).

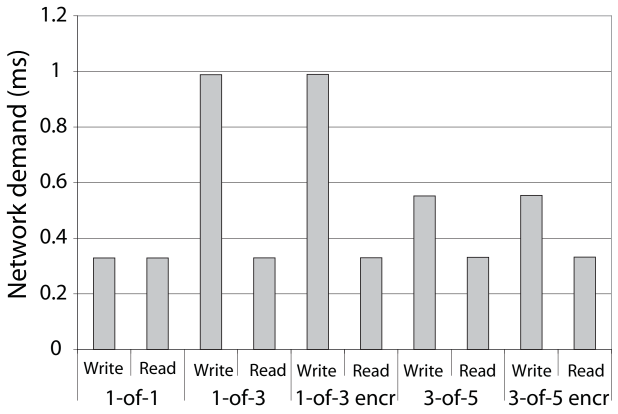

Figure 4: **Network *What*...*if* module output.** This figure illustrates how the network demand per request can differ based on the chosen encoding.

Buffer Cache *What*...*if* module: Recall from section 3.3.6 that the goal of the buffer cache module is to answer sub-questions of the form "*What* is the average number of requests $\eta_i^{I/O}$ that miss in buffer cache (and thus have to go to disk) *if* a workload from client i is added to a storage-node?". Figure5 illustrates the accuracy of the buffer cache module under three workloads of varying work-set size and access patterns. The encoding for these workloads is 1-of-1. For each of the workloads, the ATI collected the original buffer cache reference trace when the buffer cache size was 512 MB and the *What*...*if* module predicted what will happen for all other buffer cache sizes. (The choice of 512 MB is rather arbitrary, but we have verified that any other size in the range shown gives similar results). This experiment illustrates what would happen if, for example, another workload was added to the storage-node and the amount of buffer cache dedicated to the original one shrinked, or if a workload was removed from the storage-node and the amount of buffer cache dedicated to the original one was increased.

The simulator's throughput, in terms of requests that can be simulated per second is an important metric too. We have observed that for cache hits the simulator and real cache manager need similar times to process a request. The simulator is on average three orders of magnitude faster than the real system when handling cache misses (the simulator spends at most 9,500 CPU cycles in a 3.0 GhZ processor, while the real system

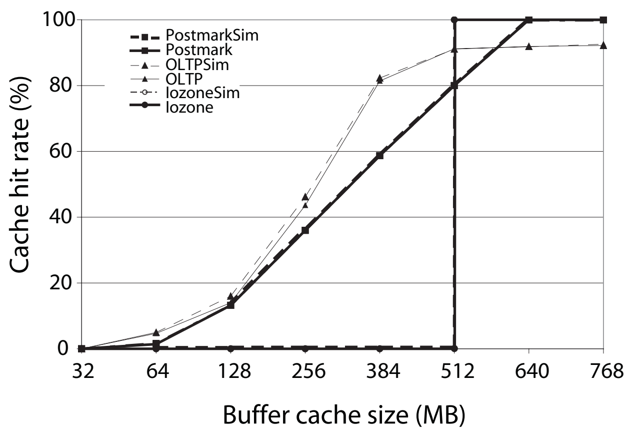

Figure 5: **Buffer Cache *What*...*if* module output.** This figure illustrates the accuracy of the buffer cache simulator in predicting the storage-node buffer cache hit rate under various workloads. For Postmark and Iozone, the actual and predicted hit rate are almost indistinguishable, indicating excellent prediction accuracy.

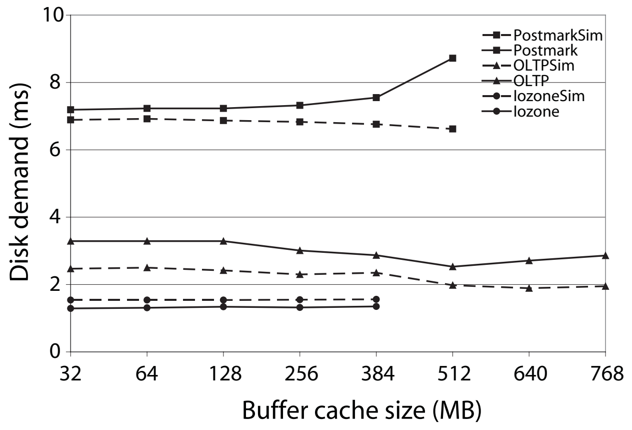

Figure 6: **Disk *What*...*if* module output.** This figure illustrates the accuracy of the disk module in predicting request service times for several workloads with different access patterns.

spends about 22,548,578 cycles)

Disk *What*...*if* module: Recall from section 3.3.7 that the goal of the disk module is to answer sub-questions of the form "*What* is the average service time $D_i^{I/O}$ of a request from client i *if* that request is part of a random/sequential read/write stream?" Figure 6 illustrates the accuracy of the disk module. The buffer cache module produces a disk reference trace (for requests that miss in buffer cache) and the disk module takes those requests, analyzes their access patterns, and predicts individual request service times from them. The module captures well the service time trends, but there is room for improvement, as seen in the Postmark case. The rather large inaccuracy at the 512 MB buffer cache size, in that case, occurs because more requests are hitting in the buffer cache, and those few requests that go to disk are serviced in FIFO fashion, thereby reducing the efficiency of the disk head in scheduling them. Our module is built assuming a full disk queue, which enables more efficient disk scheduling; in general predicting the size of the disk queue requires assumptions about arrival patterns (e.g., Poisson arrivals) that we do not want to make. The prediction inaccuracy seen is the penalty we pay for having a generic model. In practice, however, such a

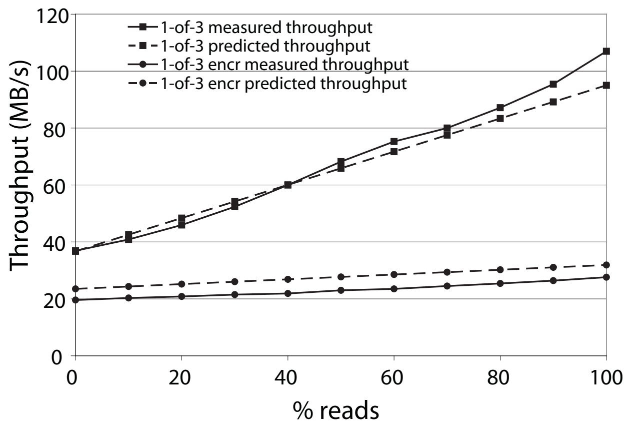

Figure 7: **Predicting maximum throughput for CPU-bound workloads.** The high-level performance question that this experiment answers is "*What* is the maximum throughput client A can get *if* its workload's encoding changes from 3-way replication to 3-way replication with encryption?"

model is sufficient to show when the disk becomes a bottleneck. More accurate disk modules, (e.g., based on simulation [Bucy03]) could be used to improve the accuracy.

4.3. Automation agent predictions

This section evaluates the accuracy of the Automation Agent in predicting the throughput and response time using several of the *What*...*if* modules in combination.

Predicting cost of encryption: The first experiment is illustrated in Figure 7. The high-level performance question that this experiment answers is "*What* is the maximum throughput client A can get *if* its workload's encoding changes from 3-way replication to 3-way replication with encryption (or the other way around)?" We answer this question considering different read:write ratios the client's workload may have. We use the SSIO_BENCHMARK to change the read:write ratio. There are several trends worth noting. First, the prediction tracks well the actual throughput lines. Second, when using encryption, the client's CPU is the bottleneck resource, hence the maximum throughput is limited by its speed. Third, as the read percentage increases, the throughput for the encoding without encryption increases, since reads obtain data from only one of the storage-nodes, while writes need to update all three storage-nodes, thus placing more load on the client's network card.

Replication vs. erasure codes: The second experiment is illustrated in Figure 8. The high-level performance question that this experiment answers is "*What* is the maximum throughput client A can get *if* its workload's encoding changes from 3-way replication to using a 3-of-5 erasure coding scheme (or the other way around)?". A 3-of-5 scheme is more storage efficient than 3-way replication, while tolerating the same number of storage-node faults (two). We answer this question for different read:write ratios using the SSIO_BENCHMARK to change the read:write ratio. The prediction accuracy for the 3-of-5 scheme is less than that of the 3-way replication. We believe this arises from a TCP inflow problem, as has also been observed in similar systems [Nagle04]. When reading under the 3-of-5 encoding, three storage-nodes are contacted to send the data. They send it to the client simultaneously, causing packet collisions on the network switch, and subsequent TCP retransmissions. We plan to incorporate this loss in throughput due to TCP retransmissions in our network module in the future.

A trend worth noting is that, for a mostly-write workload, the 3-of-5 encoding performs best, since the workloads are network bound. The amount of "extra" data that needs to be transmitted to tolerate two faults is three times more than the data that needs to be transmitted when no faults are tolerated, for the 3-way

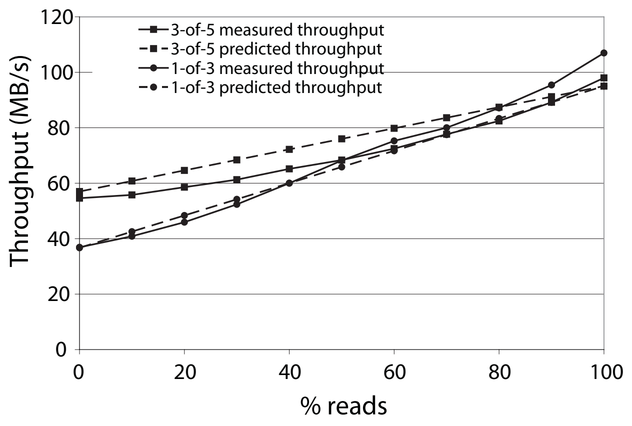

Figure 8: **Predicting maximum throughput for network-bound workloads.** The high-level performance question that this experiment answers is "*What* is the maximum throughput client A can get *if* its workload's encoding changes from 3-way replication to using a 3-of-5 scheme?"

replication; however, the 3-of-5 scheme only transmits $\frac{5}{3}$ times more data, hence the network demand is less for that scheme.

Data placement: The next experiment answers the question "*What* is the maximum throughput client A can get *if* its workload is moved to a new set of storage-nodes?" Client A's workload is encoded using 3-way replication. Two sets of possible nodes are considered for data placement. The first set S_1 currently services a second workload and the ATI measures a load of 50% on the network of the nodes of the set. The second set S_2 is currently not utilized, however one of the nodes is behind a slow 100 Mbps network (several machines in our cluster are behind this slower network and are waiting to be upgraded). Figure 9 shows that the accuracy of the predicted performance implication is reasonable.

Data placement with buffer cache interference: The next experiment is also concerned with the data placement question "*What* is the maximum throughput client B can get *if* its workload is moved to a new set of storage-nodes?", and the encoding is also 3-way replication but there are several setup differences. The first set of nodes S_1 is currently being used by a sequential workload A that hits in the buffer cache of the storage-nodes. The workload B accesses data randomly, and the administrator wants to know the performance implication of moving that workload to the S_1 set of storage-nodes. Figure 10 shows the results. The prediction is shown for both original and new setups. Several observations can be made. First, The prediction accuracy is reasonable for both workloads. Second, once workload B is added, it interferes with the buffer cache accesses of workload A, causing workload A to miss in cache. The buffer cache *What*...*if* module correctly predicts the resulting hit and miss rate for each of the workloads. Third, although workload A is inherently sequential and it should theoretically get a higher bandwidth from disk that workload B, its sequentiality is disrupted by having workload A's requests interleave with those of workload B. The disk *What*...*if* module correctly predicts the resulting service time for each of the workloads.

Throughput and response time distributions: The next experiment answers the question "*What* is the distribution of throughput and response time *if* the number of outstanding requests from client A changes?" It is intuitive that the client's throughput will peak after a certain number of outstanding requests, while the response time may continue to increase after that point as more requests are queued. Our predictive infrastructure quantifies the change in both metrics. Figure 11 illustrates the prediction accuracy for a client that is using the 3-of-5 scheme and is network-bound. After the request pipeline fills up ($N^* = 3$) the throughput peaks, while the response time increases linearly as the formulas in sections 3.3.2 and 3.3.3 predicted. The ATI monitors the actual number of outstanding requests from a client from an online system,

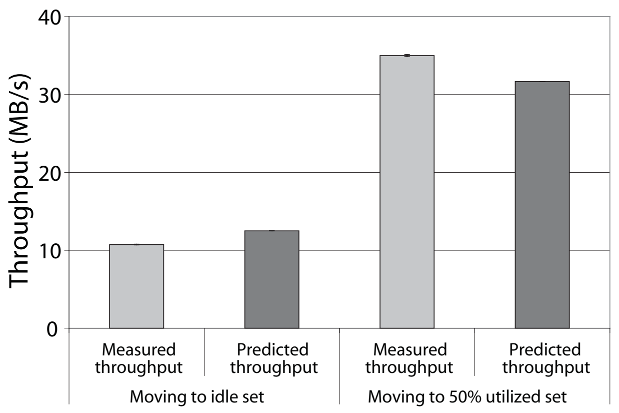

Figure 9: **Predicting maximum throughput for workload movements.** The high-level performance question that this experiment answers is "*What* is the maximum throughput client A can get *if* its workload is moved to a new set of storage-nodes?" In this experiment the first set of nodes contains a second workload that places a 50% load on the network. The second set of nodes is not loaded, however one of the machines in that set is behind a slow network.

and predicts the expected client throughput and response time. In addition it predicts the peak throughput achievable and the minimum number of outstanding requests needed to do so.

4.4. Overhead of predictive infrastructure

The predictive infrastructure is lightweight enough for common usage. There are approximately 200 instrumentation points in Ursa Minor, that post the records shown in Table 2. The ATI places demands on the CPU for encoding and decoding trace records, as well as network and storage for sending the records to the Activity DBs and storing them. It also places a fixed demand of 50 MB of buffer cache at each client computer for buffering records temporarily. The impact of the instrumentation on the above benchmarks' performance is observed to be less than 6%. The efficiency of querying the instrumentation framework for generating per-client, per-resource demands is on-par with the efficiency of databases to parse and process SQL commands. In Ursa Minor, 5% of the storage space is dedicated to keeping the activity records.

5 Related work

This section discusses related work not already addressed in the flow of the paper.

What...*if* **explorations in systems**: Some prior systems have successfully used model-driven explorations to optimize performance, especially in the area of capacity planning. Ergastulum computes a good initial configuration of a storage system by iterating over the space of possible workload characteristics and storage device models [Anderson01a]. Hippodrome builds on Ergastulum and continuously refines the configuration based on online workload-system observations [Anderson02]. We share the same goals, however we want to have system support throughout and incorporate predictive models within the system. There are differences in the systems considered too: Ursa Minor is decentralized rather than within one enclosure and it is versatile allowing for many more configuration options.

Indy [Hardwick01] identifies performance bottlenecks in a running system and attempts to predict the bottleneck shifts resulting from resource upgrade. Indy treats the system as a black box, hence the help it gets from the system is limited. Indy still requires an expert who knows what kinds of workloads the system

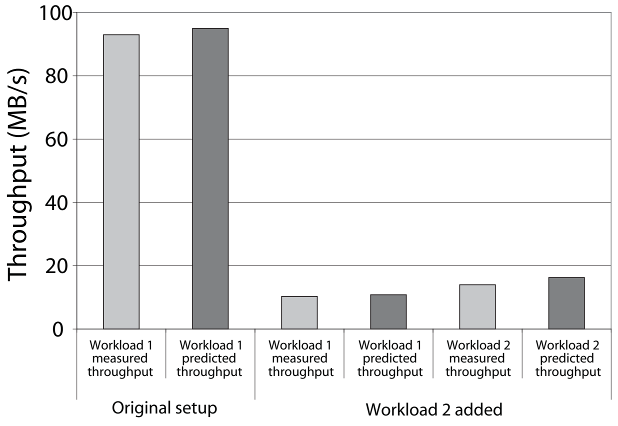

Figure 10: **Predicting maximum throughput for workload movements.** The high-level performance question that this experiment answers is "*What* is the maximum throughput client A can get *if* its workload is moved to a new set of storage-nodes?" In this experiment, the first set of nodes contains a second workload that is sequential and hits in the buffer cache.

should be able to support and who can provide required system models, all from an external view of the system. Ursa Minor has self-prediction at its core, sidestepping the need for this external expert.

What...*if* explorations have been successful for administrators of other systems other than storage. In database systems, the AutoAdmin tool can answer *What*...*if* performance questions as a function of the indices created [Chaudhuri98a]. The DB2 advisor provides similar functionality [Valentin00]. The Resource Advisor answers *What*...*if* questions related to changing the database buffer size [Narayanan05a].

Data distribution selection: Categorization of encoding schemes and their trade-offs can be found in [Chen94, Rabin89, Weatherspoon02, Wylie05, Wylie00]. We extend such work by providing a predictive framework, within the system, for choosing the right encoding based on observed system conditions and workload characteristics. AutoRAID [Wilkes96] provides versatile storage in a monolithic disk array controller. AutoRAID automatically adapts the choice for a data block (between RAID 5 and mirroring) based on usage patterns. Our system is distributed, hence we do not have a central point that monitors workloads. Our system also provides a larger spectrum of encoding choices, whereas AutoRAID provides just two.

Instrumentation frameworks and prediction algorithms: Most existing monitoring systems depend on isolated performance counters and logs that the administrator is expected to collect, filter and analyze and are designed with a single-node system in mind [Bouhana96, IBMPerf, ETW, Windows2003, OraclePerf]. Other monitoring systems scale well in distributed systems [Anderson97e, Massie04], but provide only aggregate resource consumption statistics, and do not maintain per-client information. Such aggregate performance monitors cannot differentiate among different workloads in a shared distributed system. This makes it difficult to answer finer-grained *What*...*if* questions. We designed the ATI for self-monitoring. It uses more detailed per-client activity records that keep track of all resources touched by a request as it goes through the system. In that respect, our instrumentation framework is most similar to Magpie [Barham04], a recent project at MSR.

Work has been done on pinpointing performance bottlenecks in systems. In a middleware-based system, Chen et al. show that by instrumenting just the middleware, several resource bottlenecks can be detected [Chen04h]. Aguilera et al. describe a system where software components are treated as black boxes and bottlenecks are detected by monitoring the packets flowing among them [Aguilera03b]; the instrumentation framework provides coarse-grained answers in that case. Ursa Minor has detailed instrumentation built-in and can provide finer-grained, per-client answers.

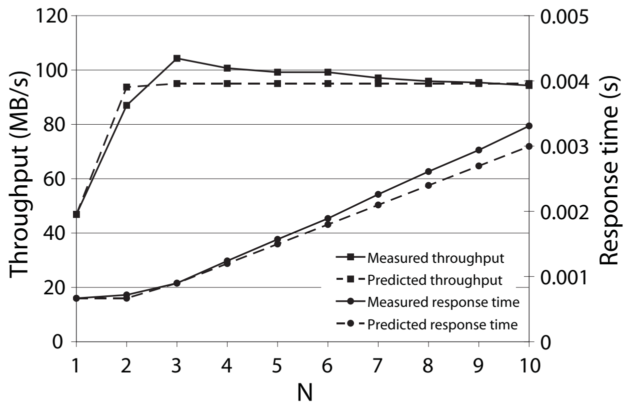

Figure 11: **Predicting throughput and response time distributions.** The high-level performance question that this experiment answers is "*What* is the distribution of throughput and response time *if* the number of outstanding requests from client A changes?"

Much research has been done on prediction algorithms. It is not our goal to invent new methods, but to rather design systems so that these approaches can work. We utilize queuing analysis to make predictions [Lazowska84a]. Other broad methods include statistical and machine learning techniques [Dinda06, Goldszmidt03].

6 Discussion and future work

There are several issues that are ongoing research. This section discusses some of them.

Re-distribution times: When making a data distribution selection, another input to the Automation Agents may be the desired upper bound on the time it takes to re-distribute the data. It is conceivable that there could be desirable data placement decisions that provide high performance, but which require a long re-distribution time to get to. An extension to our work would be predicting the re-distribution time and factoring it in when making the final decision.

Better *What*...*if* modules: There are several improvements that can be made to our *What*...*if* modules. First, as discussed throughout this paper, they make use of simple simulation or analytical models. We opted for simple models that account for first order effects. However, there is room for improvement, especially for the disk models. Second, our modules currently deal only with closed-loop workloads. An important extension of this work is to handle open-loop workloads as well. In practice, it is difficult to tell, from within the system, whether a client's workload is open- or closed-loop (or a hybrid). The client itself may give a hint to the system and that would work well. We are also investigating techniques for inferring that information from within the system, with no external hints.

Third, when workloads have distinct phases (e.g., they alternate between sequential and random reads), our system currently makes an aggregate prediction over all phases. Ideally we'd like to identify the phases automatically and make a prediction for each of them.

Fourth, there are operating regions of workload characteristics which may not lead well to accurate predictions. For example, if the block size the client uses is very small (e.g. less than 4 KB), then we have observed that a significant portion of the client and storage-node's CPU is taken by TCP processing, as illustrated in Figure 12. Such overhead could prevent a client that uses encryption, for example, from getting the full predicted throughput. Our infrastructure, should, at a minimum, identify the regions where predictions are possible, and be able to predict when not to predict.

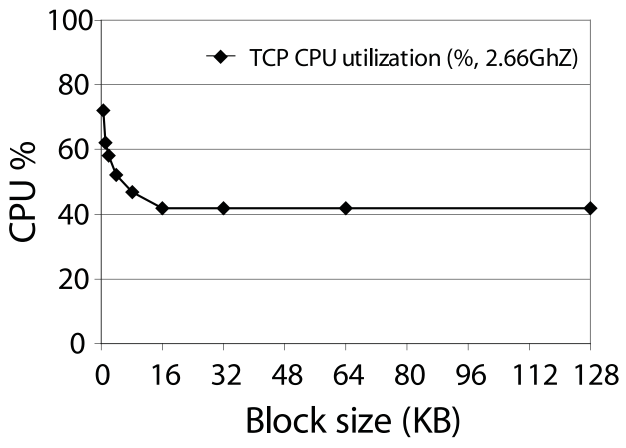

Figure 12: **TCP CPU overhead as function of the block size.**

Diverse *What*...*if* explorations: The ATI is built to help solve several performance tuning/debugging questions. In this paper we show how to predict the client's throughput and response times. It is equally interesting to predict the throughput and response times associated with a file or group of files. A database, for example, may be accessed by many clients concurrently. An interesting *What*...*if* question would be "*What* is the maximum throughput clients can get from this database, *if* the database is encoded using scheme E?". We are hoping to have a flexible ATI such that these questions can be answered with minimal effort.

Performance debugging: There could be instances where hardware and/or software mis-configurations in the system cause the system behavior to diverge from what is expected. For example, we observed several instances where switches and NICs were not configured right, and the system was not getting the performance predicted. In those cases, we believe our predictive infrastructure could still be of value to an administrator and provide suggestions for the cause of the problem. For example, a typical suggestion could be: "The client should be getting 50 MB/s of throughput, and it is only getting 20 MB/s. The CPUs and network are underutilized, and the workload is hitting in the buffer cache, so the disk is not a bottleneck. Perhaps there is a problem with the switch to the storage-nodes". Such suggestions would reduce, but not eliminate, the time the administrator needs to spend to find the root cause of the problem.

System-level support for the ATI: An assumption in designing the ATI for our storage system is that machines will only run our code and the underlying operating system. Because of this assumption, most of the activity on each computer is storage-related activities, which means that the bulk part of the ATI can reside in user space. The only modification we had to make outside our code-base was a small change in the Linux OS to allow for posting of context switch records (in Windows there is already built-in support for this [ETW]). However, there still remains the question on how to handle off-the-shelf components, like databases, which are closed source but we may still want to use in our system. Accounting for the resources such components use can only be done coarse-grained currently.

7 Summary

A self-predicting system monitors itself and answers *What*...*if* questions about hypothetical changes. This paper describes and evaluates self-prediction support in a distributed storage system that can accurately answer *What*...*if* questions about the performance impact of data encoding changes, adding or removing datasets/workloads, and adding or removing servers. The results demonstrate the feasibility of self-prediction in a real system, and we believe that the same monitoring architecture and modeling tools will work in general. Such self-prediction reduces the amount an administrator must understand about the complex workload-system interactions and is a step towards the goal of self-managing distributed systems.

References

[Abd-el-malek05b] Michael Abd-El-Malek, William V. Courtright II, Chuck Cranor, Gregory R. Ganger, James Hendricks, Andrew J. Klosterman, Michael Mesnier, Manish Prasad, Brandon Salmon, Raja R. Sambasivan, Shafeeq Sinnamohideen, John D. Strunk, Eno Thereska, Matthew Wachs, and Jay J. Wylie. Ursa Minor: versatile cluster-based storage. *Conference on File and Storage Technologies* (San Francisco, CA, 13–16 December 2005), pages 59–72. USENIX Association, 2005.

[Aguilera03b] Marcos K. Aguilera, Jeffrey C. Mogul, Janet L. Wiener, Patrick Reynolds, and Athicha Muthitacharoen. Performance debugging for distributed systems of black boxes. *ACM Symposium on Operating System Principles* (Bolton Landing, NY, 19–22 October 2003), pages 74–89. ACM Press, 2003.

[Anderson01a] Eric Anderson, Mahesh Kallahalla, Susan Spence, Ram Swaminathan, and Qian Wang. *Ergastulum: an approach to solving the workload and device configuration problem*. Technical report HPL–SSP–2001–05. HP Labs, 2001.

[Anderson02] Eric Anderson, Michael Hobbs, Kimberly Keeton, Susan Spence, Mustafa Uysal, and Alistair Veitch. Hippodrome: running circles around storage administration. *Conference on File and Storage Technologies* (Monterey, CA, 28–30 January 2002), pages 175–188. USENIX Association, 2002.

[Anderson97e] Eric Anderson and Dave Patterson. Extensible, scalable monitoring for clusters of computers. *Systems Administration Conference* (San Diego, CA, 26–31 October 1997), pages 9–16. USENIX Association, 1997.

[Barham04] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using Magpie for request extraction and workload modelling. *Symposium on Operating Systems Design and Implementation* (San Francisco, CA, December 2004), 2004.

[Bouhana96] James P. Bouhana. UNIX Workload Characterization Using Process Accounting. *22nd International Computer Measurement Group Conference* (San Diego, CA, 10–13 December 1996), pages 379–390, 1996.

[Bucy03] John S. Bucy and Gregory R. Ganger. *The DiskSim simulation environment version 3.0 reference manual*. Technical Report CMU–CS–03–102. Department of Computer Science Carnegie-Mellon University, Pittsburgh, PA, January 2003.

[Chaudhuri98a] Surajit Chaudhuri and Vivek Narasayya. AutoAdmin what–if index analysis utility. *ACM SIGMOD International Conference on Management of Data* (Seattle, WA, 01–04 June 1998), pages 367–378. ACM Press, 1998.

[Chen04h] Mike Y. Chen, Anthony Accardi, Emre Kiciman, Dave Patterson, Armando Fox, and Eric Brewer. Path-based failure and evolution management. *Symposium on networked system design and implementation* (San Francisco, CA, 29–31 March 2004), pages 309–322, 2004.

[Chen94] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. RAID: high-performance, reliable secondary storage. *ACM Computing Surveys*, **26**(2):145–185, June 1994.

[Dinda06] Peter A. Dinda. Design, implementation, and performance of an extensible toolkit for resource prediction in distributed systems. *IEEE Transactions on Parallel and Distributed Systems,*, **17**(2):160–173. IEEE, February 2006.

[ETW] Microsoft. Event tracing, 2005. http://msdn.microsoft.com/.

[Ganger03a] Gregory R. Ganger, John D. Strunk, and Andrew J. Klosterman. *Self-* Storage: brick-based storage with automated administration*. Technical Report CMU–CS–03–178. Carnegie Mellon University, August 2003.

[Gartner00] Gartner Group. Total Cost of Storage Ownership — A User-oriented Approach, February, 2000. Research note, Gartner Group.

[Goldszmidt03] Moises Goldszmidt and Bikash Sabata. Research issues in applying pattern reconfiguration and statistical models to system managment and modeling. *Algorithms and Architectures for Self-Managing Systems* (San Diego, CA, 11–11 June 2003), pages 29–34. HP Labs, IET Inc., 2003.

[Goodson04] Garth R. Goodson, Jay J. Wylie, Gregory R. Ganger, and Michael K. Reiter. Efficient Byzantine-tolerant erasure-coded storage. *International Conference on Dependable Systems and Networks* (Florence, Italy, 28 June – 01 July 2004), 2004.

[Gray03] Jim Gray. A conversation with Jim Gray. *ACM Queue*, **1**(4). ACM, June 2003.

[Hardwick01] Jonathan Hardwick, Efstathios Papaefstathiou, and David Guimbellot. Modeling the Performance of E-Commerce Sites. *27th International Conference of the Computer Measurement Group* (Anaheim, CA, 2001). Published as *Journal of Computer Resource Management*, **105:3**(12), 2001.

[IBMPerf] IBM. DB2 Performance Expert, 2004. http://www-306.ibm.com/software.

[Katcher97] Jeffrey Katcher. *PostMark: a new file system benchmark*. Technical report TR3022. Network Appliance, October 1997.

[Keeton04] Kimberley Keeton, Cipriano Santos, Dirk Beyer, Jeffrey Chase, and John Wilkes. Designing for disasters. *Conference on File and Storage Technologies* (San Francisco, CA, 31 March–02 April 2004), pages 59–72. USENIX Association, 2004.

[Lazowska84a] E. Lazowska, J. Zahorjan, S. Graham, and K. Sevcik. *Quantitative system performance: computer system analysis using queuing network models*. Prentice Hall, 1984.

[Massie04] Matthew L. Massie, Brent N. Chun, and David E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, **30**(7), July 2004.

[Nagle04] David Nagle, Denis Serenyi, and Abbie Matthews. The Panasas ActiveScale storage cluster - delivering scalable high bandwidth storage. *ACM Symposium on Theory of Computing* (Pittsburgh, PA, 06–12 November 2004), 2004.

[Narayanan05a] Dushyanth Narayanan, Eno Thereska, and Anastassia Ailamaki. Continuous resource monitoring for self-predicting DBMS. *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)* (Atlanta, GA, 27–29 September 2005), 2005.

[Norcott02] William Norcott and Don Capps. Iozone filesystem benchmark program, 2002. http://www.iozone.org.

[OraclePerf] Oracle. Oracle Database Manageability, 2004. http://www.oracle.com/technology/.

[Rabin89] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, **36**(2):335–348. ACM, April 1989.

[Saito04a] Yasushi Saito, Svend Frølund, Alistair Veitch, Arif Merchant, and Susan Spence. FAB: building distributed enterprise disk arrays from commodity components. *Architectural Support for Programming Languages and Operating Systems* (Boston, MA, 09–13 October 2004), pages 48–58. ACM, 2004.

[Soules03] Craig A. N. Soules, Garth R. Goodson, John D. Strunk, and Gregory R. Ganger. Metadata efficiency in versioning file systems. *Conference on File and Storage Technologies* (San Francisco, CA, 31 March–02 April 2003), pages 43–58. USENIX Association, 2003.

[TPCCdoc] Transaction Processing Performance Council. TPC Benchmark C, December 2002. http://www.tpc.org/tpcc/Revision5.1.0.

[Valentin00] Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy M. Lohman, and Alan Skelley. DB2 Advisor: An optimizer smart enough to recommend its own indexes. *International Conference on Data Engineering* (San Diego, CA, 28–03 February 2000), pages 101–110, 2000.

[Weatherspoon02] Hakim Weatherspoon and John D. Kubiatowicz. Erasure coding vs. replication: a quantitative approach. *International Workshop on Peer-to-Peer Systems* (Cambridge, MA, 07–08 March 2002). Springer-Verlag, 2002.

[Wilkes96] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems*, **14**(1):108–136, February 1996.

[Windows2003] Microsoft. Windows Server 2003 Performance Counters Reference, 2005. http://www.microsoft.com/technet/.

[Wylie00] Jay J. Wylie, Michael W. Bigrigg, John D. Strunk, Gregory R. Ganger, Han Kiliccote, and Pradeep K. Khosla. Survivable information storage systems. *IEEE Computer*, **33**(8):61–68. IEEE, August 2000.

[Wylie05] Jay J. Wylie. *A read/write protocol family for versatile storage infrastructures*. PhD thesis, published as PhD Thesis CMU-PDL-05-108. Carnegie Mellon University, October 2005.

[Zhang04b] Zheng Zhang, Shiding Lin, Qiao Lian, and Chao Jin. RepStore: a self-managing and self-tuning storage backend with smart bricks. *International Conference on Autonomic Computing* (New York, NY, 17–18 May 2004), pages 122–129. IEEE, 2004.