Proceedings of the International Conference on Autonomic Computing (ICAC-06),
Dublin, Ireland. June 12th-16th 2006.

Informed data distribution selection in a self-predicting storage system

Eno Thereska*, Michael Abd-El-Malek*, Jay J. Wylie†, Dushyanth Narayanan‡, Gregory R. Ganger*
*Carnegie Mellon University, Pittsburgh, PA
†HP Labs, Palo Alto, CA
‡Microsoft Research, Cambridge, UK

***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' decision making, such as acquisition planning and performance tuning, 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.**

I. Introduction

Storage administration is a difficult and expensive task [1], [2], [3]. One major portion of this task involves making decisions about such things as acquisitions, component configurations, assignment of datasets/workloads to components, and performance problem resolutions. For many of these decisions, the most complex aspect is understanding the performance consequences of any given decision.[1] These consequences usually depend on workload specifics (e.g., the interleaved I/O patterns of the applications) and storage system internals.

Traditionally, administrators use two tools when making such decisions: their expertise and system overprovisioning [4]. 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, overprovisioning 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.

[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.

We believe that systems must provide better assistance to administrators. 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 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 guiding, planning and tuning 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 [5], 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 impact of any given data distribution.

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 relative to replication. Data placement requires similar knowledge plus information about 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 such 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. Aggregate performance counters typically exposed by systems are insufficient for this purpose. Our system uses end-to-end instrumentation in the form of traces of "activity records" that mark steps reached in the processing of any given request. Those traces are post-processed to compute demands and latencies. Modules for answering *What*...*if* questions use modeling tools and 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?" needs answers to questions about how the cache hit rate, disk workload and network utilization would change.

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, and the answers 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.

II. 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 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 have developed a generic infrastructure that can evaluate the impact of hypothetical choices.

A. Cluster-based storage systems

Traditional storage systems are built around a single-vendor, monolithic disk array design. Such systems provide high performance and availability, but they are expensive and do not scale easily. Incremental scaling is not an option with such systems, as a client must buy and manage another monolithic system when scalability requirements slightly exceed the existing array system. Cluster-based storage systems, built from commodity hardware, have been developed to address these scalability and cost issues [5], [6], [7]. The individual servers are often called *storage-nodes*, and each provides a certain amount of CPU, buffer cache and storage. These components can be inexpensive mass-produced commodities. 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 the face of node failures. A data distribution is an algorithm for encoding the data to meet availability and confidentiality needs and choosing the set of storage-nodes to host the data.

TABLE I
Data encoding tunable parameters

n	Data is encoded in n fragments.
m	Any m fragments reconstruct the data.
encryption type	Encryption ensures confidentiality.

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 [8], and thus 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 may require that response times be kept below a threshold.

B. 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 [9], [10]. Most erasure coding schemes can be characterized by the parameters (m, n). An m-of-n scheme encodes data into n *fragments* such that reading any m of them reconstructs the original data. Confidentiality requirements dictate whether or not encryption is employed. Encryption is performed prior to encoding (and decryption is performed after decoding). Table I 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 reads. 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 III-C.4 and Section III-C.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. As the evaluation section shows, 3-way replication (i.e., a 1-of-3 encoding) consumes approximately 40% more network bandwidth than 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 [11], [12].

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. This paper shows that a system can 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* data is encoded using scheme E?".

C. Data placement

In addition to selecting the data encoding, the storage-nodes on which encoded data fragments are placed must also be selected. When data is initially created, the question of placement must be answered. Afterwards, different system events may cause 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 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 existing 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.

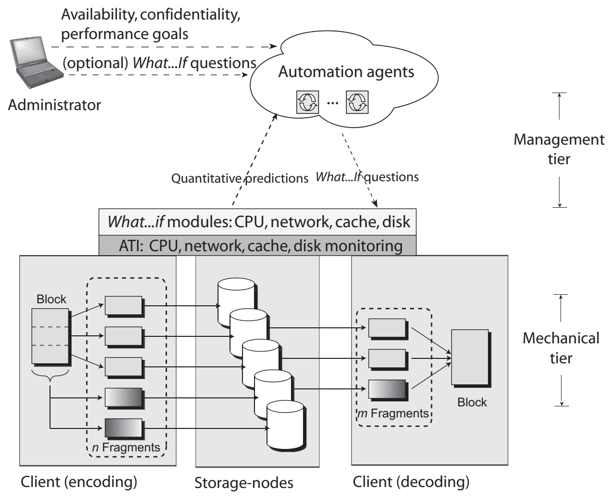

Fig. 1. **High-level architecture of Ursa Minor.** The mechanical tier, on the bottom, services I/O requests for clients. The management tier, on the top, provides automation. It makes use of the self-predicting capabilities of the individual system components to get answers to various *What*...*if*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

III. System design and architecture

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

A. 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. [5]. At the core of its architecture is the separation of mechanical functions (servicing client requests) from management functions (automating administrative activities). The management 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 Ursa Minor 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 use the PASIS consistency protocols to read and write data [12], [13]. Clients include an Ursa Minor library that encodes data and implements the PASIS protocols. Illustrated in Figure 1 are two clients. The first is writing data using a 3-of-5 encoding scheme (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, a buffer cache and disks. Storage-nodes are expected to be heterogeneous, as they get upgraded or retired over time and are purchased from different vendors.

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 [8]). Confidentiality may be specified by the data encryption method. Performance goals are often expressed in terms of service-level objectives 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. Automation agents come with a set of pre-defined *What...if* questions that the system implementor has made available. Previous work has shown how to convert availability and confidentiality goals into encoding decisions [8], [12], [13]. This paper focuses 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 Ursa Minor. The ATI is integrated in every storage-node and the Ursa Minor client library. The ATI presents a unified distributed performance monitoring infrastructure and allows differentiation among multiple workload streams. *What...if* modules use the ATI to quantify resource consumption by different clients and make performance predictions regarding hypothetical workload and/or resource changes. Predictions from several resource-specific *What...if* modules may be analyzed by the automation agents to make high-level throughput and response time predictions. The ATI design and implementation details are described by Thereska et al. [14]. The next section summarizes key features of the ATI used to make informed data distribution decisions.

B. Activity tracking infrastructure (ATI)

The ATI is responsible for tracking the performance of every client request along its execution path. The ATI retains activity records, such as buffer cache reference records, I/O records, and network transmit/receive records. The sequence of records allows 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 simulation techniques to answer *What...if* questions.

An activity record is a sequence of (attribute, value) pairs. Figure 2 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. Each timestamp is a unique value generated by the computer clock 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. Activity records are posted on the critical path; however, as our evaluation shows,

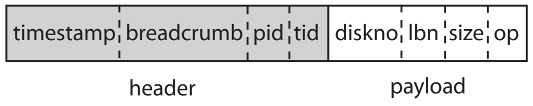

Fig. 2. **Example activity record.** Each activity record has a common header and a payload. The payload for the disk request activity record shown includes the disk id, logical block number, size of the I/O, and operation type.

such posting causes minimal impact on foreground performance. Table II lists the instrumentation points in Ursa Minor. *KernelProcessSwitch* records are provided by the Linux kernel[3]; the other records are posted from instrumentation points in user-level processes. There are approximately 200 instrumentation points in Ursa Minor.

Each computer runs a single ATI instance. An 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 an 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. Storing activity records in a database permits efficient execution of queries.

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. Activity records are effectively a super-set of performance counters. Any performance counter value of interest can be extracted by querying the DBs.

C. *What...if* modules

What...if modules are structured in two tiers. The lower tier answers performance questions pertaining to individual 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.

1) Performance metrics: Performance metrics of interest in this paper are expected client *throughput* and *response time* under a hypothetical data distribution change. In addition, we want to predict client *peak achievable throughput*. Throughput depends on the number of outstanding requests the client issues to fill the pipeline of request processing. 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. Other metrics such as throughput and response time variance are also important, but such second-order statistics are difficult to predict. They require making assumptions about workload characteristics (e.g., exponential interarrival times) that in practice may or may not hold.

[3]Other operating systems, such as Windows, also expose kernel-level context switches [15].

TABLE II

ACTIVITY RECORD TYPES POSTED IN URSA MINOR

	Record Type	Arguments	Description
CPU demand	*UserThreadSwitch*	*oldthread, newthread*	A user-level context switch
	KernelProcessSwitch	*cpu ID, oldprocess, newprocess*	A kernel-level 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 and marks buffer dirty
	BufferReadAhead	*file, offset, numpages, pagesize*	Prefetch pages (non-blocking)
	BufferFlush	*file, offset, size*	Flushes a dirty page to disk
	BufferEvict	*file, offset, size*	Evicts a page from the cache
Network demand	*NetworkTransmit*	*sender, receiver, numbytes*	Monitors network flow
Disk demand	*DiskOp*	*disk ID, LBN, size, operation*	Monitors disk activity

2) Throughput prediction: To predict aggregate throughput under a hypothetical distribution change, our algorithms assume a closed-loop workload[4] and use operational analysis [16] on all resources (CPUs, networks, buffer cache, disks). Table III is a reference for the notation used.

TABLE III

FORMULA NOTATION

D_i^k	Average demand of a request from client i on resource k.
D_i^{max}	Largest demand client i places on any resource.
D_i	Sum of demands on all resources client i's request uses.
N_i	Average number of requests a client has outstanding.
N_i^*	Threshold for determining if client i's load is low or high.
p_i	Buffer cache hit rate for client i.
R_i	Response time bound for client i.
T_i	Throughput bound for client i.
Z_i	Average client think time.

Let D_i^k be the average demand, in seconds, of a request from client i on resource k. 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). Let D_i be the sum of all demands on all resources a request uses.

If the ATI measures that client i has, on average, N_i requests outstanding, and that the average client think time (or processing time after a request is received and before the next is sent) is Z_i, then client i's throughput bound T_i is:

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

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 ($N_i/(D_i + Z_i)$). 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 $N_i^* = (D_i + Z_i)/D_i^{max}$, where N_i^* can be thought of as the minimum number of requests required to keep the request pipeline full.

The peak throughput of the client CPU, in terms of requests it can process, 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 III-C.4.

The peak network throughput, in terms of number of requests it can process, 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 runs. The new network demand is predicted based on the observed workload patterns and the new configuration as described in Section III-C.5.

The peak storage-node throughput, in terms of number of requests that it can process, equals $1/D_i^{I/O}$, where $D_i^{I/O}$ equals $p_i * D_i^{BUF} + (1 - p_i) * D_i^{DISK}$. Some read requests hit in the buffer cache (with probability p_i) and their service time is the access time from the cache. The other read requests miss in the buffer cache (with probability $1 - p_i$) and incur a service time denoted by D_i^{DISK}. All write requests eventually go to disk, hence p_i for them is always zero.

[4]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 p_i and D_i^{DISK} need to be predicted for a hypothetical data distribution. Both depend heavily on the interaction among the node's buffer cache size and eviction policy and disks at each node, as explained in Sections III-C.6 and III-C.7 respectively. They also depend on workload access patterns (e.g., sequential or random).

3) Response time prediction: We predict response time R_i by transforming our throughput predictions above using Little's law [16], which states that

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

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.

4) CPU $\underline{What}\ldots\underline{if}$ module: The goal of the client CPU module[5] is to answer sub-questions of the form "$\underline{What}$ is the request demand D_i^{CPU} for requests from client i $\underline{if}$ the data is 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

[5]There is CPU consumed at the storage-nodes as well (e.g., during data copying). 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.

each encode/decode operation is short in duration. Direct measurements sidestep the need for constructing analytical models for different CPU architectures. 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 block several times with the new hypothetical encoding and produces the average CPU demand 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 more 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. Furthermore, as discussed in the extended technical report of this paper [17], some CPU is also consumed by network TCP processing. Currently, for every machine in the system, we construct a simple TCP CPU consumption model by measuring the CPU consumed when the network link is fully saturated, using different client request sizes.

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 the data is 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 retrieves 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 an empirical fixed cost for the network stack processing. The time to transmit data equals the size of the request in bytes divided by the network bandwidth. The fragment's size is the original request block size divided by m.

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 fraction of read requests $1-p_i$ that miss in the 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 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 size and replacement policy.

Consider workload W_w being placed on a storage-node that already hosts $w-1$ workloads $W_1,...,W_{w-1}$. The prediction takes the form:

$$\{p_1,p_2,...,p_w\} = BufferCache_{module}\{W_1,W_2,...,W_w\} \quad (3)$$

The buffer cache module uses simulation to make a prediction. The module uses buffer cache records of each of the $W_1,W_2,...,W_w$ workloads, collected through the ATI, and replays them using the buffer cache size and policies of the target storage-node. The output from this module is the fraction of hits and misses and a trace of requests that have to go to disk for each workload.

Simulation is used, rather than an analytical model, 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.

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^{DISK} 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 [18].

When a disk is installed, a model is built for it. The model is based on the disks maximum random read and write bandwidth and maximum sequential read and write bandwidth. These four parameters are usually provided by disk vendors and are also easy to extract empirically.

The disk module is analytical. It receives the interleaved sequence of I/Os of the different workloads from the buffer cache What...if module, scans the combined trace to find sequential and random streams within it, and assigns an expected service time to each request, based on the four extracted disk parameters.

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 peak 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.

IV. 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 regarding 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.

A. Experimental setup

The experiments use a cluster of x86-based computers. Clients are run on 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; each has a single Intel 82546 gigabit Ethernet adapter, and they are 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 PASIS access protocol. Micro-benchmarks access the storage-nodes directly using the PASIS access protocol.

SSIO_BENCHMARK: This micro-benchmark allows control of the workload read:write ratio, access patterns and number of outstanding requests. The performance of a 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 mimics an on-line database performing transaction processing. Transactions invoke 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. 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 measure streaming data access (e.g., for data mining) [19]. For our experiments, IOzone measures 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 six data encodings. These results are indicative of the many other encodings we explored. "1-of-1" refers to 1-way replication. "1-of-1 encr" is 1-way replication where the data is also encrypted to ensure confidentiality. For encryption, we use the AES cipher with a key size of 128 bits in CBC mode. "1-of-3" is 3-way replication which tolerates two storage-node crashes. "1-of-3 encr" is 3-way replication with encryption. "3-of-5" is an erasure coding scheme that tolerates two storage-node crashes, 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 and the standard deviation are reported. The average client think time Z_i is zero in all experiments.

B. Resource-specific What...if modules

This section evaluates 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. For those two resources we just show how resource consumption changes as a function of encoding choice. The memory and disk What...if modules are based on simulation and analytical models, respectively, and so we concentrate on the prediction accuracy of these modules.

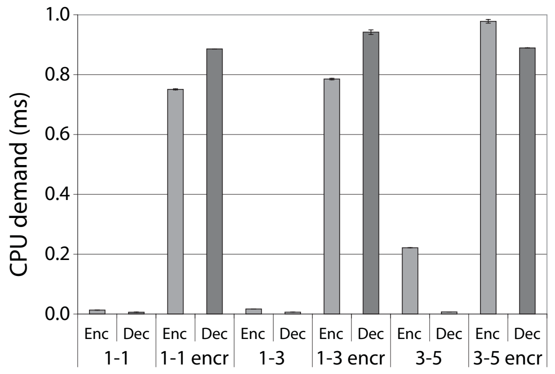

Fig. 3. **CPU What...if module output.** This figure illustrates how the CPU demand per request differs based on the chosen data encoding. The cost for encoding data (during a write) and decoding it (during a read) are shown for six encoding choices.

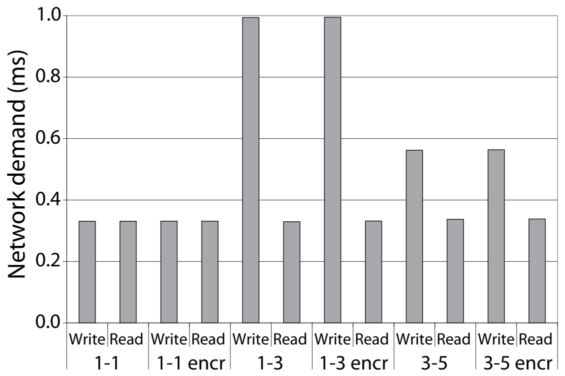

Fig. 4. **Network What...if module output.** This figure illustrates the network demand per request as a function of the chosen data encoding.

CPU What...if module: Recall from Section III-C.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 the data is encoded using scheme E?". Figure 3 shows how the CPU demand varies based on the encoding scheme used. The module runs 100 encode/decode operations 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 can become a bottleneck. Two trends are worth noting. First, the cost of encryption dominates the cost of data encoding/decoding. Second, erasure coding requires more CPU than replication for encoding data.

Network What...if module: Recall from Section III-C.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 the data is encoded using scheme E?". Figure 4 shows how the network demand varies based on the encoding schemes used. A trend worth noting, is that replication places a larger demand on the network than erasure coding, for the same number of storage-node crashes tolerated. In general,

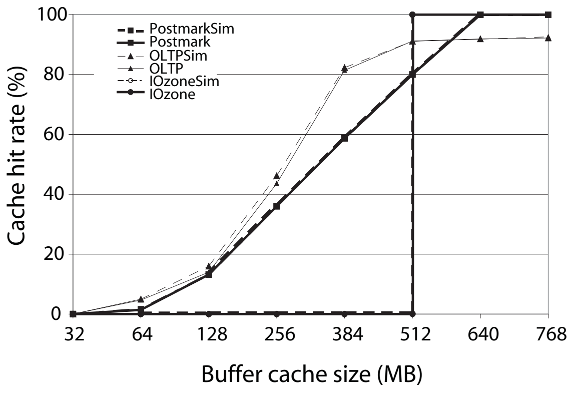

Fig. 5. **Buffer Cache *What*...*if* module output.** This figure illustrates the accuracy of the buffer cache simulator in predicting the storage-node buffer cache hit rate under various workloads. For Postmark and IOzone, the measured and predicted hit rate are almost indistinguishable, indicating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

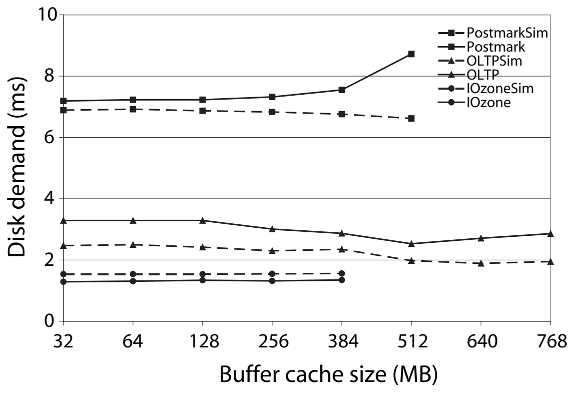

Fig. 6. **Disk *What*...*if* module output.** This figure illustrates the accuracy of the disk module in predicting request service times for several workloads with different access patterns.

for n-way replication, $n * blocksize$ bytes are sent through the network during a write. For m-of-n erasure coding schemes, on the other hand, only $n * blocksize/m$ bytes are sent. During reads, both schemes consume the same amount of network bandwidth.

Buffer Cache *What*...*if* module: Recall from Section III-C.6 that the goal of the buffer cache module is to answer sub-questions of the form "*What* is the average fraction of read requests $1 - p_i$ that miss in the buffer cache (and thus have to go to disk) *if* a workload from client i is added to a storage-node?". Figure 5 illustrates the accuracy of the buffer cache module under three workloads of varying working-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 available to the original one shrank, or if a workload was removed from the storage-node and the amount of buffer cache available to the original one increased.

An important metric for evaluating the efficiency of the buffer cache *What*...*if* module is the simulator's throughput, in terms of requests that can be simulated per second. 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 handling a miss, whereas, on a 3.0 Ghz processor, the real system spends the equivalent of about 22,500,000 CPU cycles).

Disk *What*...*if* module: Recall from Section III-C.7 that the goal of the disk module is to answer sub-questions of the form "*What* is the average service time D_i^{DISK} 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. 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 occurs because more requests are hitting in the buffer cache, and the few requests that go to disk are serviced in FIFO fashion, thereby reducing the efficiency of the disk head scheduler. Recall from Section III-C.7 that the disk module is built using four parameters extracted form the disk: maximum random read and write bandwidth and maximum sequential read and write bandwidth. Maximum bandwidths usually imply that the disk queue size is full (that leads to more efficient disk scheduling). Hence, our disk module assumes that the disk queues are always full when making a prediction. That leads to inaccuracies when the disk queue is not full (as is the case for the Postmark case). In general, predicting the size of the disk queue requires assumptions about arrival patterns (e.g., Poisson arrivals) that we do not wish to make. The prediction inaccuracy seen is the penalty we pay for using a simple model. In practice, however, such a model is sufficient in predicting when the disk becomes a bottleneck. More accurate disk modules (e.g., based on simulation) could be used to improve the accuracy.

C. *Automation agent predictions*

This section evaluates the accuracy of the automation agents 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 peak 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)?". There are several trends worth noting. First, the predictions track well the actual throughput lines. Second,

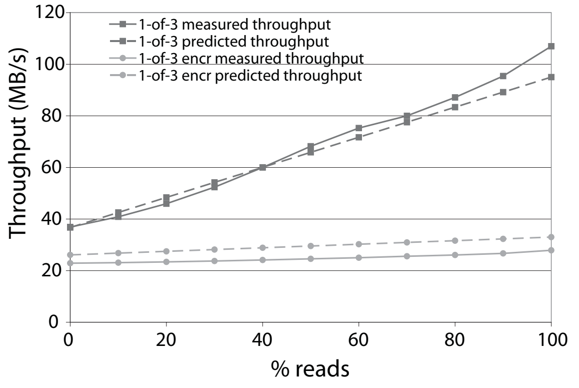

Fig. 7. **Predicting peak throughput for CPU-bound workloads.** SSIO_BENCHMARK is used to measure throughput for different read:write ratios.

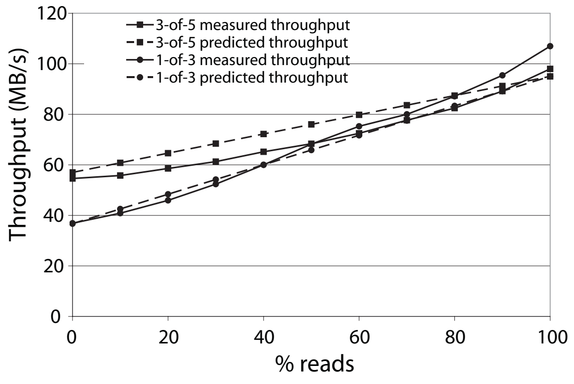

Fig. 8. **Predicting peak throughput for network-bound workloads.** SSIO_BENCHMARK is used to measure throughput for different read:write ratios.

when using encryption, the client's CPU is the bottleneck resource. Third, although the CPU cost of encoding is higher than that of decoding for the encoding with encryption, the throughput increases slightly as the read percentage increases. This is because writes are sent to three machines, thus requiring more network bandwidth than reads. As described in Section III-C.4, the higher the network bandwidth required, the higher the CPU demand needed for TCP processing. Thus, less CPU time is available for the encoding and encryption of data. Fourth, 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 network and CPU.

Replication vs. erasure codes: The second experiment is illustrated in Figure 8. The high-level performance question that this experiment answers is "<u>*What*</u> is the peak throughput client A can get <u>*if*</u> its workload's encoding changes from 3-way replication to 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 crashes (two). 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 been observed in similar systems [20]. When reading under the 3-of-5 encoding, three storage-nodes are contacted to retrieve the data. The storage-nodes simultaneously reply to the client, causing packets to be dropped at the network switch. That leads to 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 crashes is three times more than the data that needs to be transmitted when no crashes are tolerated, for the 3-way replication; however, the 3-of-5 scheme only transmits $\frac{5}{3}$ times more data. Hence, the network demand is less for that scheme.

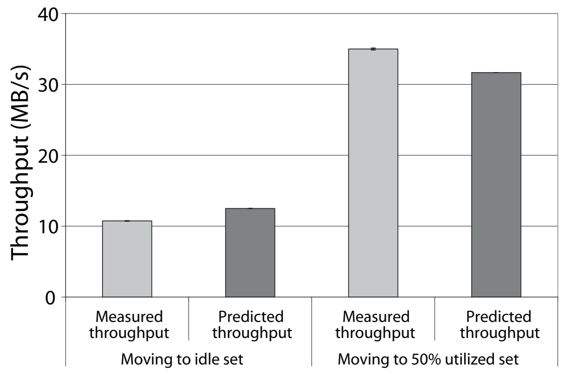

Fig. 9. **Predicting peak throughput for workload movements.** The high-level performance question that this experiment answers is "<u>*What*</u> is the peak throughput client A can get <u>*if*</u> its workload is moved to a new set of storage-nodes?" In this experiment, the first set of nodes is not loaded, however one of the machines in that set is behind a slow network. The second set of nodes contains a workload that places a 50% load on the network. Both SSIO_BENCHMARK workloads consist entirely of writes.

Data placement: The next experiment answers the question "<u>*What*</u> is the peak throughput client A can get <u>*if*</u> 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 is currently not utilized. However, one of the nodes is behind a 100 Mbps network (the other nodes are behind a 1000 Mbps, or gigabit, network). The second set S_2 currently services a second workload, and the ATI measures a load of 50% on the network of the set S_2 nodes. Figure 9 shows the accuracy of the predicted performance.

Data placement with buffer cache interference: The next experiment is also concerned with the data placement question "<u>*What*</u> is the peak throughput client B can get <u>*if*</u> its workload is moved to a new set of storage-nodes?". The encoding is again 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. Workload B accesses data randomly, and the administrator wants to know the performance implication of moving that

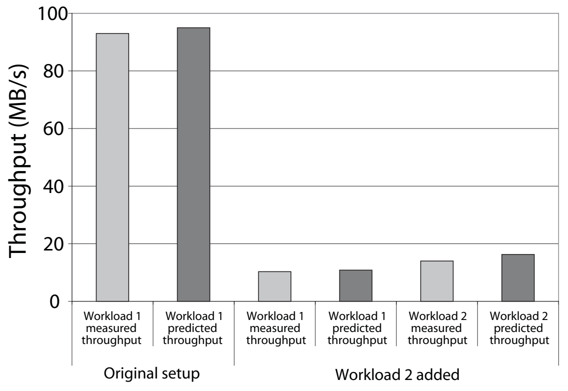

Fig. 10. **Predicting peak throughput for workload movements.** The high-level performance question that this experiment answers is "*What* is the peak throughput client A can get *if* its workload is moved to a new set of storage-nodes?" In this experiment, the first set of nodes contains a second workload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

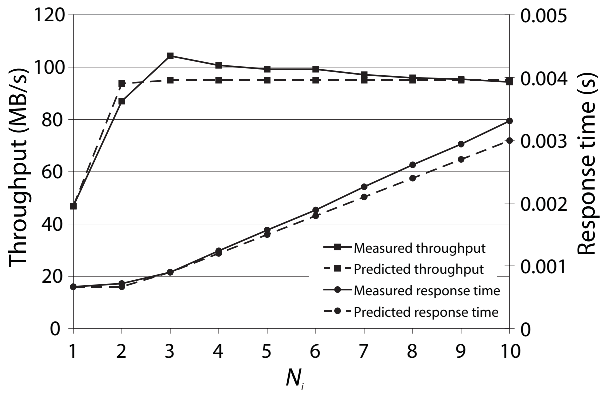

Fig. 11. **Predicting throughput and response time distributions.** The high-level performance question that this experiment answers is "*What* is the distribution of throughput and response time *if* the number of outstanding requests N_i from client i changes?" SSIO_BENCHMARK is used to control the number of outstanding requests. The workload consists entirely of reads.

workload to the S_1 set of storage-nodes. Figure 10 shows the results. The prediction is shown for both the 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 competes 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 than workload B, its requests are interleaved with those of workload B, resulting in semi-random accesses from the disk's perspective. 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 N_i from client i 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 ($N_i^* = 3$) the throughput peaks, while the response time increases linearly as the formulas in Sections III-C.2 and III-C.3 predict. The ATI monitors the actual number of outstanding requests from a client from an online system, 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.

D. 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 II. The ATI places demands on the CPU for encoding and decoding trace records, as well as network and storage demand for sending the records to the Activity DBs. It also places a fixed demand of 20 MB of buffer cache at each computer for temporarily buffering records. 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. For more details and extended evaluation of the ATI's overhead on system performance, please refer to [14].

V. Related work

***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 [21]. Hippodrome builds on Ergastulum and continuously refines the configuration based on online workload-system observations [22]. We share the same goals, but 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 [23] 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 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 database systems. The AutoAdmin tool can answer *What...if* performance questions as a function of the indices created [24]. The DB2 advisor provides similar functionality [25]. The Resource Advisor answers *What...if* questions related to changing the database buffer size [26].

Data distribution selection: Categorization of encoding schemes and their trade-offs can be found in [9], [10], [11], [12], [27]. 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 [28] 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 deals with a larger spectrum of encoding choices, whereas AutoRAID utilizes 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 [15], [29], [30], [31], [32], [33]. Other monitoring systems scale well in distributed systems [34], [35], 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 fine-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 [36].

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 [37]. Aguilera et al. describe a system where software components are treated as black-box and bottlenecks are detected by monitoring the packets flowing among them [38]; 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.

Much research has been done on prediction algorithms. It is not our goal to invent new methods, but rather to design systems so that these approaches can work. We utilize queuing analysis to make predictions [16], because we have access to the system's source code and are thus able to monitor all queues where request processing happens. Others have used statistical techniques [39], [40] to make coarse-grained performance predictions in black-box systems.

VI. Discussion

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. There is room for improvement, especially for the disk models. Second, the cost of re-distribution is not included in the models currently. It is desirable for the Automation Agents to predict the time it will take to re-distribute the data or to take as input an upper bound on how long re-distribution is allowed to take. Third, our modules currently deal only with closed-loop workloads. An important extension of this work is to predict the response time of open-loop workloads [41]. In practice, it is difficult to determine, from within the system, whether a client's workload is open- or closed-loop (or a hybrid).

An assumption in designing the ATI for Ursa Minor is that machines will only run our code and the underlying operating system. 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 [15]). It will be valuable to make predictions about off-the-shelf components, like databases, which are closed source, that store data in Ursa Minor. We expect that the accounting for the resources such components use will be coarse-grained compared to the ATI instrumentation built into Ursa Minor and so will lead to less accurate predictions.

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 correctly, 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's throughput should be 50 MB/s, but 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 network switch". Such suggestions would reduce, but not eliminate, the time the administrator needs to spend to find the root cause of a problem.

VII. 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 storage-nodes. 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.

VIII. Acknowledgements

We thank Matthew Wachs for answering questions on the Linux kernel. We thank the anonymous reviewers for their useful feedback. We thank the members and companies of the PDL Consortium (including APC, EMC, Equallogic, Hewlett-Packard, Hitachi, IBM, Intel, Microsoft, Network Appliance, Oracle, Panasas, Seagate and Sun) for their interest, insights, feedback, and support. This work is supported in part by Army Research Office grant number DAAD19-02-1-0389, by NSF grant number CNS-0326453, by DARPA's SRS program via Air Force contract number FA8750-04-01-0238, and by the Air Force Research Laboratory via contract F49620-01-1-0433.

References

[1] G. R. Ganger, J. D. Strunk, and A. J. Klosterman, "Self-* Storage: brick-based storage with automated administration," Carnegie Mellon University, Tech. Rep., August 2003.

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

[3] J. Gray, "A conversation with Jim Gray," *ACM Queue*, vol. 1, no. 4, June 2003.

[4] G. Weikum, A. Mönkeberg, C. Hasse, and P. Zabback, "Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering." in *VLDB*, August, 2002, pp. 20–31.

[5] M. Abd-El-Malek, W. V. Courtright II, C. Cranor, G. R. Ganger, J. Hendricks, A. J. Klosterman, M. Mesnier, M. Prasad, B. Salmon, R. R. Sambasivan, S. Sinnamohideen, J. D. Strunk, E. Thereska, M. Wachs, and J. J. Wylie, "Ursa Minor: versatile cluster-based storage," in *Conference on File and Storage Technologies*. USENIX Association, 2005, pp. 59–72.

[6] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence, "FAB: building distributed enterprise disk arrays from commodity components," in *Architectural Support for Programming Languages and Operating Systems*. ACM, 2004, pp. 48–58.

[7] Z. Zhang, S. Lin, Q. Lian, and C. Jin, "RepStore: a self-managing and self-tuning storage backend with smart bricks," in *International Conference on Autonomic Computing*. IEEE, 2004, pp. 122–129.

[8] K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes, "Designing for disasters," in *Conference on File and Storage Technologies*. USENIX Association, 2004, pp. 59–72.

[9] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: high-performance, reliable secondary storage," *ACM Computing Surveys*, vol. 26, no. 2, pp. 145–185, June 1994.

[10] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *Journal of the ACM*, vol. 36, no. 2, pp. 335–348, April 1989.

[11] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: a quantitative approach," in *International Workshop on Peer-to-Peer Systems*. Springer-Verlag, 2002.

[12] J. J. Wylie, "A read/write protocol family for versatile storage infrastructures," Ph.D. dissertation, Carnegie Mellon University, October 2005.

[13] G. R. Goodson, J. J. Wylie, G. R. Ganger, and M. K. Reiter, "Efficient Byzantine-tolerant erasure-coded storage," in *International Conference on Dependable Systems and Networks*, 2004.

[14] E. Thereska, B. Salmon, J. Strunk, M. Wachs, M. Abd-El-Malek, J. Lopez, and G. R. Ganger, "Stardust: Tracking activity in a distributed storage system," in *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, June, 2006.

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

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

[17] E. Thereska, M. Abd-El-Malek, J. J. Wylie, D. Narayanan, and G. R. Ganger, "Informed data distribution selection in a self-predicting storage system," Carnegie Mellon University, Tech. Rep., January 2006.

[18] C. A. N. Soules, G. R. Goodson, J. D. Strunk, and G. R. Ganger, "Metadata efficiency in versioning file systems," in *Conference on File and Storage Technologies*. USENIX Association, 2003, pp. 43–58.

[19] W. Norcott and D. Capps, "IOzone filesystem benchmark program," 2002, http://www.iozone.org.

[20] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas ActiveScale storage cluster - delivering scalable high bandwidth storage," in *ACM/IEEE SC2004*, 2004.

[21] E. Anderson, M. Kallahalla, S. Spence, R. Swaminathan, and Q. Wang, "Ergastulum: quickly finding near-optimal storage system designs," HP Labs, Tech. Rep., 2001.

[22] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch, "Hippodrome: running circles around storage administration," in *Conference on File and Storage Technologies*. USENIX Association, 2002, pp. 175–188.

[23] J. Hardwick, E. Papaefstathiou, and D. Guimbellot, "Modeling the Performance of E-Commerce Sites," in *27th International Conference of the Computer Measurement Group*, vol. 105:3, no. 12, 2001.

[24] S. Chaudhuri and V. Narasayya, "AutoAdmin what–if index analysis utility," in *ACM SIGMOD International Conference on Management of Data*. ACM Press, 1998, pp. 367–378.

[25] G. Valentin, M. Zuliani, D. C. Zilio, G. M. Lohman, and A. Skelley, "DB2 Advisor: An optimizer smart enough to recommend its own indexes," in *International Conference on Data Engineering*, 2000, pp. 101–110.

[26] D. Narayanan, E. Thereska, and A. Ailamaki, "Continuous resource monitoring for self-predicting DBMS," in *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2005.

[27] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliccote, and P. K. Khosla, "Survivable information storage systems," *IEEE Computer*, vol. 33, no. 8, pp. 61–68, August 2000.

[28] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan, "The HP AutoRAID hierarchical storage system," *ACM Transactions on Computer Systems*, vol. 14, no. 1, pp. 108–136, February 1996.

[29] B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal, "Dynamic instrumentation of production systems," in *USENIX Annual Technical Conference*. USENIX Association, 2004, pp. 15–28.

[30] J. P. Bouhana, "UNIX Workload Characterization Using Process Accounting," in *22nd International Computer Measurement Group Conference*, 1996, pp. 379–390.

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

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

[33] Oracle Corporation, "Oracle Database Manageability," 2004, http://www.oracle.com/technology/.

[34] E. Anderson and D. Patterson, "Extensible, scalable monitoring for clusters of computers," in *Systems Administration Conference*. USENIX Association, 1997, pp. 9–16.

[35] M. L. Massie, B. N. Chun, and D. E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing*, vol. 30, no. 7, July 2004.

[36] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using Magpie for request extraction and workload modelling," in *Symposium on Operating Systems Design and Implementation*, 2004.

[37] M. Y. Chen, A. Accardi, E. Kiciman, D. Patterson, A. Fox, and E. Brewer, "Path-based failure and evolution management," in *Symposium on networked system design and implementation*, 2004, pp. 309–322.

[38] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance debugging for distributed systems of black boxes," in *ACM Symposium on Operating System Principles*. ACM Press, 2003, pp. 74–89.

[39] M. Goldszmidt and B. Sabata, "Research issues in applying pattern reconfiguration and statistical models to system managment and modeling," in *Algorithms and Architectures for Self-Managing Systems*. HP Labs, IET Inc., 2003, pp. 29–34.

[40] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger, "Storage Device Performance Prediction with CART Models," in *International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*. IEEE/ACM, 2004.

[41] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Closed versus open system models and their impact on performance," in *Symposium on Networked Systems Design and Implementation*, 2006.