Data Center Workload Monitoring, Analysis, and Emulation

Justin Moore[†] Jeff Chase[†] Keith Farkas[‡] Parthasarathy Ranganathan[‡]
[†]*Department of Computer Science*
Duke University
{*justin,chase*}*@duke.edu*
[‡]*Internet Systems and Storage Lab*
Hewlett Packard Labs
{*keith.farkas, partha.ranganathan*}*@hp.com*

***Abstract*— Over the last ten years we have witnessed a shift from large mainframe computing to commodity, off-the-shelf clusters of servers. Today's data centers contain thousands or tens of thousands of servers, providing services and computation for tens or hundreds of thousands of users. In addition to traditional IT challenges such as server management, security, and performance, data center owners now must deal with power and thermal issues, previously the domain of facilities management. These trends will continue to accelerate as organizations acquire bladed servers and consolidate multiple, smaller clusters into centrally-located data centers. However, in spite of these trends, there has been no corresponding change in emphasis in the methods and toolkits that target system instrumentation, analysis, management, replay, and emulation.**

This paper seeks to address this gap. We focus on methods and toolkits to enable the automated collection and analysis of workload traces from data centers, and use those traces as the basis for repeatable and verifiable experiments and workload emulation. Our work has two components:

- **a location- and environment-aware extended knowledge plane that places thermal and power management concerns at the same level as service performance, collecting and analyzing facilities and performance data with particular focus on causal relationships across this boundary, and**
- **data analysis and and workload playback methods that allow detailed and flexible emulation of enterprise-class workloads.**

We discuss the high-level architectural requirements for these two components and present results from specific implementations and toolkits.

I. Introduction

As data centers grow in size and proliferate, we have seen a wide range of applications evolve to take advantage of this environment. Web farms with multiple tiers, multimedia rendering applications, large-scale simulations, and other service-oriented workloads now scale to tens of thousands of servers. This new world presents challenges to both the owners of these data centers and the customers or users who run the applications. Data center owners must manage facilities-level resources — such as the power grid and computer room air conditioning (CRAC) units — in addition to traditional information technology (IT) level resources. Users must manage applications that may be run on shared hardware, including virtual machines and virtual local area networks, and in heterogeneous environments.

This work is supported in part by HP Labs, the U.S. National Science Foundation (EIA-9972879 and EIA-9870728), and an IBM faculty research award.

The scale of this challenge has motivated recent work in frameworks for coordinated monitoring and control of large-scale computing infrastructures. The most common approaches are based on *M*onitor, *A*nalyze, *P*lan, and *E*xecute (MAPE) control loops. Figure 1 provides a high-level overview of these projects. An instrumentation infrastructure logs sensor readings, which undergo data analysis. The results of the analysis are fed to a policy engine, which creates a plan for how to utilize resources. Finally, external interfaces to data center objects allow the administrator — or other actors — to monitor the data center and react to changing conditions from remote locations in a rapid manner. For example, commercial frameworks such as HP's OpenView and IBM's Tivoli aggregate information from a variety of sources and present a graphical monitoring and control interface to administrators.

Recent research focuses on extending the state of the art in three significant ways. The first is to extend it to Internet-scale systems, often using a sensor metaphor for the instrumentation, and leveraging research in large-scale sensor networks [11] and queries on massive sensor fields [17], [8] for wide-area infrastructures such as PlanetLab [18] or the Grid [6]. The second is to develop analysis tools to recognize patterns and diagnose anomalies in the data [4], [3], [1], [10]. Finally, since human operators may be unable to assess events quickly enough to respond effectively, there is increasing interest in "closing the loop" with tools to plan responses, and execute them through programmatic interfaces (actuators) for system management; for example, this is a key long-term goal of initiatives for autonomic computing and adaptive enterprises at IBM and HP respectively. These trends combine in the idea of a "knowledge plane" for the Internet and other large-scale systems [5].

Physical information has an important role to play in dynamic monitoring and control for data center automation as well, particularly when coupled with performance metrics. As a motivating example, consider the need to manage power and cooling in a data center. The cost of energy to power and cool the equipment of a large data center is significant (e.g., $72,000 per month for a 40,000 sq. ft. facility[12]). Moreover, technology trends are driving increasing power density, in part to reduce costs for space and cabling. As a result, the infrastructure for power and cooling is critical

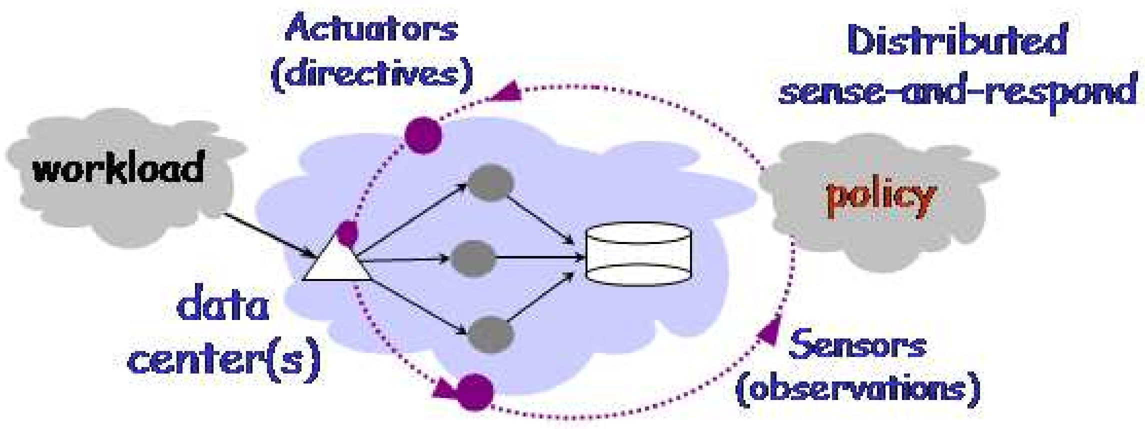


Fig. 1. High-level view of modern data center operation. Workload —in the form of web requests, data analysis, multimedia rendering, or other applications – is placed in the data center. An instrumentation infrastructure monitors server activity, network utilization, and power and temperature status. A policy actor, either an administrator or management software, analyzes the observations, and formulates a management plan. The actor leverages external control interfaces, or *actuators*, to implement the plan.

to reliability, and automated mechanisms to control power consumption and cooling resources are essential. Combined instrumentation is a prerequisite for intelligent control to adjust a software-controlled cooling infrastructure [14], place workloads to balance thermal load and improve energy efficiency, or forecast thermal events and respond by migrating or throttling workloads or failing over to backup systems.

A fundamental challenge facing these projects, however, is how to conduct effective scientific experiments that provide insight into these new environments. There are a shortage of methods and tools to obtain data and understand the interactions between objects in the data center — from the low-level facilities components to the high-level application performance — and then validate or reject hypotheses on how these components will respond to future changes and optimizations. If we are to apply the scientific method to data center management, we must have the right tools to perform repeatable, verifiable experiments, and measure the results.

This work addresses a portion of this problem by presenting three tools that focus on data collection, analysis, and workload emulation. *Splice*, our data collection tool, enables us to correlate observations across the IT/facilities boundaries and understand the location-dependent aspects of data center management, such as the temperature throughout the data center. Our data analysis tool, *SeASR*, helps us understand how objects respond and change during experiments, and provides feedback to *Splice*, enabling more efficient data collection and retention. The final application, *sstress*, enables fine-grained and repeatable control over server resource utilization, allowing us to explore the IT/facilities relationships in one machine, or emulate workload playback across the data center.

The rest of this paper is organized as follows. Section II examines the requirements for our toolkit and the high-level architecture for each. Section III describes the implementation of each tool and the assumptions behind our decisions. Section IV presents some results and experiments performed to examine the facilities/IT boundary, and Section V concludes.

II. Architecture

This section describes our approach to data center instrumentation, statistical analysis, and workload emulation. The instrumentation and analysis architectures also serve as the basis for an automated, closed-loop control component, or *site authority* [2].

A. Monitor

The design of Splice was guided by two emerging data center trends. First, data centers are increasingly dynamic. New equipment is continually added and existing equipment is often reconfigured or removed; both kinds of changes may include adjustments to the supporting power, cooling, and network infrastructure. Similarly, each successive generation of equipment offers new capabilities and new features. Second, with the drive towards larger data centers and consolidation, the number of measurement points and objects within data centers continues to grow.

These trends drive the following goals for the Splice data model:

1) It must be extensible to support new data sources, new objects, and new object properties.
2) It must archive a history of changes to the state of the data center over time, although most queries apply to the current state. In this context, state includes the objects that comprise the data center, their location and other properties, how they are composited, and their infrastructure connection points. As such, Splice must enable an agent to retrieve the state of the data center at any time in the past.
3) The architecture must be scalable to large numbers of objects and attributes, and long histories.

The data collection and filtering engine logs changing values of dynamic attributes for environmental and performance metrics, such as CPU utilization, power draw, and CPU temperature for a server, along with other information that defines the state of the data center.

Splice gathers data from many sources that may publish data through different interfaces and with different formats. The data collection engine includes modules that implement the required elements of the communication interface associated with each such data source. For example, we have built a communication module to interface the engine to both the Ganglia and OpenView Service Reporter performance tools, as well as to temperature and power sensors using proprietary interfaces and OPC (OLE for Process Control), a standard that is widely used in commercial sensors for process control and manufacturing automation.

Along with each such data item, the communication interfaces also gather a time stamp corresponding to when the item was recorded. Many sensors timestamp data at the source; for example, some sensor interfaces cache data readings locally until the aggregator "pulls" it using operations for polling and retrieval. In our current implementation, we rely on the Network Time Protocol (NTP) for clock synchronization of server-hosted sensors. For sensors that produce data at regular intervals, Splice timestamps each reading locally before entering it in the database.

A second role performed by the data collection and filtering engine is to filter the incoming data streams or filter the values already recorded in the database. Filtering reduces the amount of data that is stored in the database, improving scalability. The amount of data impacts the speed at which data can be inserted into the database, and to a greater degree, the speed at which data can be retrieved.

Splice uses a change-of-value filter that retains only those values that differ significantly from the previously logged values; this reduces data size significantly, with minimal loss of information. Splice also supports a variation of this approach in which a more aggressive filter is applied to older data thereby trading increased information loss for greater compression ratios. The filter parameters are defined on a per-data-source basis. Some sensors may also filter continuous readings before publishing values for a measurement interval.

B. *Analyze*

With the instrumentation infrastructure providing attribute data, the next step is data analysis. Our data analysis falls into two main classes: attribute behavior, and correlation. Attribute behavior describes both the value of observed readings, as well as how those values change over time. Hidden Markov Models (HMMs) are useful for this purpose, as they can summarize both the distribution of observations and how the values will change. Attribute behavior analysis can provide data conduits in the instrumentation infrastructure with information on how to establish filtering policies. For example, analysis may indicate that a majority of consecutive CPU load average readings differ by less than 12%, with a sharp drop-off in the distribution above that threshold; therefore, a filtering policy that discards a reading that differs from the previously logged reading by 12% will preserve a similar degree of accuracy as a different cutoff value (for example, an "intuitive" value of 10%), but will reduce necessary storage space significantly.

Data correlation methods determine which attributes affect other attributes and the strength of those correlations. This analysis is necessary to determine the minimum set of control points necessary to control an indirectly managed attribute. In our context, correlation is a relatively lightweight process that determines access rights for our external control points; correlation will not tell us how to control the attributes. For example, our correlation methods will let us know that ambient temperature is a function of server CPU utilization, fan speeds, CRAC supply temperature, and floor vents, but is not a factor of server power consumption; servers consume power spinning disks, but do not release the power as heat.

C. *Emulation*

There is a growing body of work that focuses on emulation. As opposed to simulation — in which all aspects of the system are modeled — emulation allows one to leverage real hardware and applications, and simulate only portions of the world. Similar to other tools, such as ModelNet [16] and Emulab [19], we seek to emphasize the "science" in computer science; it is important to conduct repeatable tests on live systems to quantify the behavior of the systems we design.

The desired functionality is the ability to take a sequence of CPU, memory, disk, and network utilization figures for one or more servers and force another set of servers to recreate those conditions. Fine-grained control over these four attributes also enables us to perform detailed analysis of a single server, including the relationship between utilization and power consumption. On a larger scale, we wish to be able to analyze the effect of different workload placement algorithms on IT and facilities-level components, including request queue sizes and CRAC cooling costs.

III. Implementation

This section describes the implementation of the three portions of our workload monitoring, analysis, and emulation toolkit.

A. *Splice*

Splice aggregates sensor and performance data in a relational database using a database schema that has been designed to treat information that rarely changes in much the same way as those that frequently change. That is, the schema uses the same set of tables to store information that rarely changes, such as the physical memory size of a system and the power circuit to which it is attached, and information that frequently changes, such as the power consumption and CPU utilization of the system. In so doing, the schema addresses our two extensibility and adaptability goals. However, there are two exceptions. First, we assume that the size of objects is immutable, and hence, this information is stored as part of the objects definition. Second, we track separately the current and past location of objects so as to reduce the time required to access this important parameter.

Turning to specifics, the database schema comprises eight tables, which are illustrated in Figure 2. The *object types* table records the basic information about the types of object in the data center, while the *objects* table records the instances

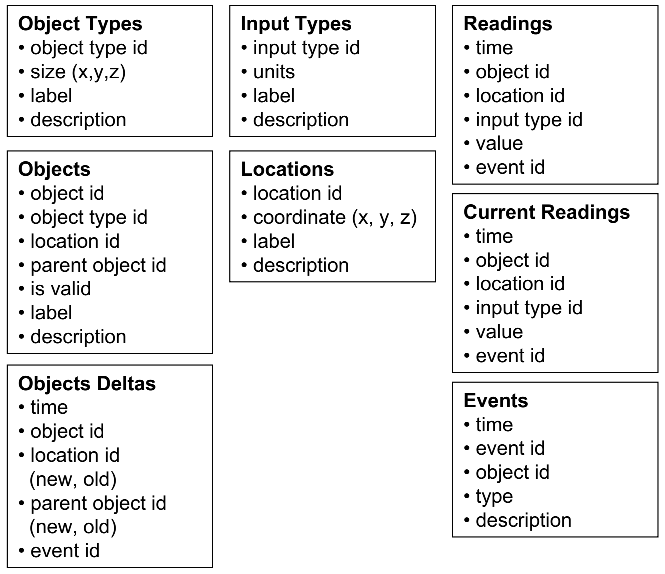


Fig. 2. Database schema of the Splice architecture.

of each object type, its parent object identifier, its location identifier, and whether it is currently present in the data center. The parent object identifier is used to specify an "attached-to" relationship between two objects, such as, that a system is a part of a rack, or a power grid connection point connects to a particular system. If an object is moved, its earlier location and "attach-to" relationship is recorded in the *object deltas* table before the new location is recorded into the objects table.

With the exception of object size and location, the *readings* table records all the properties of the objects, both, as noted above, those that are dynamic and those that are static. For each reading, this table records the object that provided the reading (e.g., a power meter, a temperature sensor), the location identifier of the object, the input-type identifier of the reading, and the reading value. The location identifier is included so as to support objects that are mobile, and thus, the object identifier alone is not sufficient to locate the reading. The input-type identifier keys into the *input types* table, which provides the units for the reading along with a description and label. These items are useful to agents. Finally, the *current readings* table records the latest reading for each property. A separate table is provided to reduce the time required to extract the current value of all properties, and hence, facilitates agents that require real-time access to the information, such as monitoring functions or control systems.

Each of the above mentioned tables records a location identifier rather than spatial coordinates. This approach was chosen to reduce the amount of duplicate information in the database. Location identifiers are mapped to spatial coordinates by the *locations* table. The frame of reference for the spatial coordinates is a top-level object, namely, the data center. Multiple data centers may be supported by defining a non-overlapping region of 3D space for each.

Finally, the *events* table allows a management agent to log an user-defined event. Event types may be stand-alone occurrences (e.g., a new system was installed), or may mark the start and end of a sequence (e.g., the cooling system was down for a day). As such, the event table provides context for the other information maintained in the database.

B. SeASR

We created the *Se*nsor *A*nalysis and *S*ynthetic *R*eproduction toolkit (SeASR) to provide us with a way to discover important statistical properties of observed sensor readings. Using these properties, we can create synthetic traces for use in various experiments. SeASR was written with Splice in mind – particularly Splice's filtering capabilities – and includes features that allow us to analyze and reproduce data stored in a Splice database. SeASR examines how readings change, how often sensors update readings, and into what range of values attribute readings fall.

The statistical properties of how attribute values change are crucial to the effectiveness of Splice's filters, and allows both administrators and automated components to set optimal per-attribute filtering parameters. Simple properties, such as the mean and standard deviation, are of limited use; one trace whose attribute values change seldom but with great variation may have an identical mean and deviation as another trace whose values change often but with little variation. The same logic applies to how often the values change. Given these observations, we make two assumptions.

- First, we assert that the observed distribution of attribute value deltas and time deltas — the time between consecutive readings — for any one attribute type is each the sum of one or more Gaussian distributions. In addition to the observed distributions, we create a pseudo-distribution having a delta of all zeros. Thus we have N "states" the delta can be in: $N-1$ observed Gaussian distributions and the all-zeros distribution.
- Second, we assume that there is are correlations between the N states. For example, the attribute whose values change seldom but with great variation will have a high probability that one zero delta will follow another zero delta. We create an NxN first-order Hidden Markov Model to predict the odds that one distribution will follow another.

SeASR uses one-dimensional Expectation Maximization (EM) clustering, a common technique in machine learning, to create the Gaussian distributions that serve as the basis for the HMM. We use these distributions as input for the forward-backward algorithm, which produces the Hidden Markov Model [15]. Using the set of Gaussian distributions and the HMM, we use our *mktrace* tool to create a synthetic trace of sensor readings. The user can specify location, attribute, and object identifiers for each trace, as well as the start and end times of the trace. These statistics and synthetic traces are useful in scalability tests, setting filtering parameters, and detecting unusual behavior in future readings.

C. Sstress

Sstress is an application for selectively utilizing parts of a single machine or networked servers. It is a multi-threaded application, accepting commands from stdin and starting, changing, or stopping worker threads. Sstress currently supports four classes of worker threads: CPU, RAM, disk, and network. On startup, sstress will attempt to determine the effective speed of the system, including CPU speed, random number generator speed, and system call overhead. This burn-in process will run for a few seconds and then allow the user to enter commands. There are three commands — add, mod, del — to control thread execution. When adding a thread, you can specify the attributes of the new worker thread.

We can specify the number of concurrent processor loads we want to run. While we can't explicitly assign load to each CPU of an SMP system, we can have multiple threads running with load, and hopefully the scheduler will be smart enough to distribute the load in an intelligent manner. The worker threads use alternating periods of executing "add" instructions in a tight loop with periods of sleeping by using the SELECT system call. Currently there are 100 "slots" per second, and sstress will perform one "add" cycle per percent of CPU time requested.

We specify the number of megabytes of RAM we want to use. We do this by asking for the total megabytes of RAM to allocate, the megabytes of the working set, and the delay (in microseconds) before "touching" each page in the working set. Sstress attempts to keep memory in RAM and out of swap by performing a random walk through the working set. This feature is not 100% effective, as it depends on the scheduler. The MLOCK (memory lock) function is one possibility, but requires root access; we currently try to avoid that requirement.

Sstress can establish multiple I/O streams to read and write data from a given file. For each stream, we can specify the filename, the blocksize in kilobytes, and the number of I/Os per second. Sstress can also establish multiple I/O streams over the network. For each stream, we can specify the address and port of the remote end, the transport (UDP or TCP), the read/write mode, the packet size, and the I/O rate in packets per second.

A networked version off sstress – netsstress – and a perl script that emits sstress commands enables data center-wide trace replay with these different policies. We used sstress extensively in preliminary stages of our temperature-aware workload placement research.

IV. Results

This section examines the effectiveness of our instrumentation and analysis components, and the flexibility of our emulation toolkit.

A. *Instrumentation and Analysis*

Here we examine the results of running Splice on two clusters: HP's Utility Data Center (UDC) and the Duke Computer Science "Devil Cluster". The amount of data arriving at the database from the conduits has the potential to be very large; after only six weeks the Duke database has over thirteen millions rows, consuming 800 MB in data and index files. Complex queries over large data sets need to scale gracefully if the data is to be useful. For a single Splice site to attain such scalability we explore the benefits of simple data filtering techniques. For example, we can often discard consecutive identical or near-identical readings from a given sensor. In this manner, we explore *delta-value filtering* and *age-delta value filtering*.

In delta-value filtering we take a value as it arrives at the database and compare it to the current reading for that sensor. If difference between the new value and the old value is less than some threshold, we discard the new value. If they are sufficiently different, however, we add the new value to the database. The amount a value is allowed to change before we log a new entry is the *size* of the filter. We trade perfect accuracy – defined as logging every reading arriving at the database – for scalability. Splice allows us to establish per-conduit and per-sensor-type filtering policies, irregardless of any filtering that may or may not occur on the other end of the conduit.

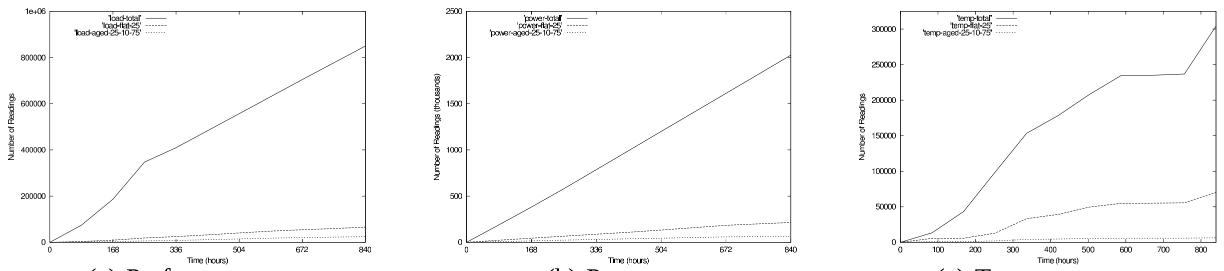
Age-delta value filtering is similar to delta-value filtering, but with the addition of a postprocessing filter. The postprocessor examines historical data and allows us to increase the size of the filter for older data. The rationale behind this approach is that as data gets older the exact values become less important. However, to ensure that old data does not indiscriminately get filtered out, there is an upper bound on the coarseness of the filtering granularity.

1) Exploring Filtering: Filtering is a tradeoff between accuracy and space, and here we explore the costs and benefits of various filtering parameters.

Figure 3 illustrates the effect of these filtering modes for three kinds of sensors: power, temperature, and one-minute CPU load. The CPU data (Figure 3(a)) is presented for the Duke facility while the power (Figure 3(b)) and temperature (Figure 3(c)) data is presented for the HPL facility.

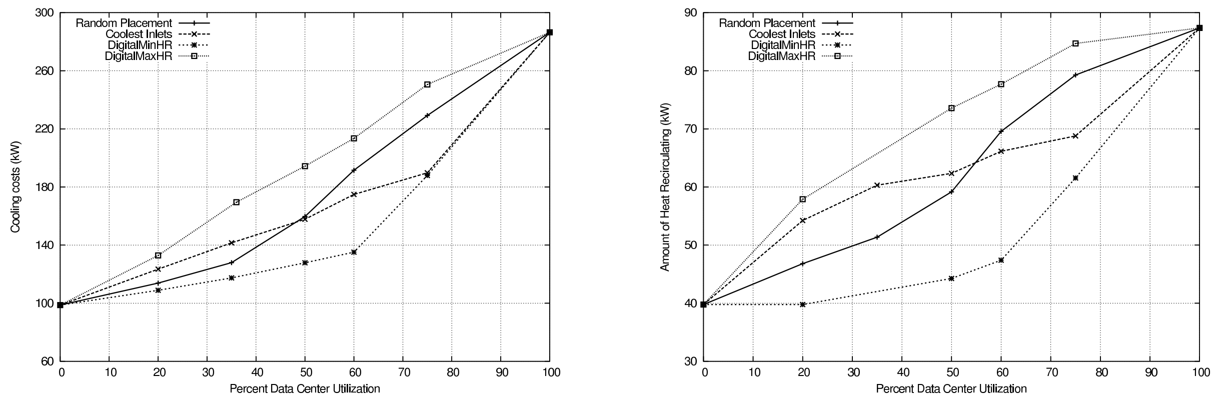
For each graph, the line illustrating the no-filtering default case shows a large increase in the number of readings over a five-week period when compared to simple delta-value filtering. For performance data that is logged close to every two minutes, such as the one-minute load average, this is almost a factor of fourteen; for the base unfiltered graphs for the temperature and power data, the increases are factors of five and ten respectively. Even one sensor that logs one reading every other minute will generate over nine megabytes of data per year, not including database metadata; one rack of machines at the HP UDC can easily generate over a terabyte of measurements.

Focusing on Figures 3(a) and (b), the results indicate that the delta-value filter is very effective at reducing the database size. For the five-week total, this method reduces the number of readings by almost 85% for the CPU load (using a delta-value-threshold of 0.25 in the one-minute CPU load-average) and 60% for the power readings (using a delta-value-threshold of 10W). As evident from the slopes of the curves, the compression ratio is fairly constant indicating that there is not too much variation for these metrics in the sample data set we considered. Figure 3 also shows the compression possible with the *aged-value-delta* approach. As expected, as time progresses, the aged-value-delta approach accomplishes better compression compared to the base delta-value method by



(a) Performance (b) Power (c) Temperature

Fig. 3. Demonstrating the filtering modes in Splice.



(a) Cooling costs for random placement, choosing the coolest inlets, and best and worst heat-recirculation-based algorithms.

(b) The amount of heat recirculating. Note that the increase in heat recirculation closely mirrors the increase in cooling costs.

Fig. 4. At mid-range utilizations, cooling costs for a data center using temperature-aware workload placement algorithms are 30% less than random workload placement and almost 40% less than the worst possible workload distribution.

virtue of its greater compression of older values.

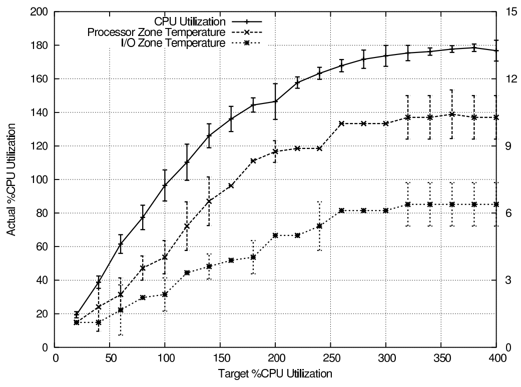
Figure 3(c) represents an interesting case where the data collection agent already includes some non-trivial amount of base pre-filtering. The OPC server logs temperature data only when the temperature difference exceeds 0.05 degrees F or if more than a half-hour has elapsed. Also, the collected-data-graph shows other variations in the slope due to periods when the temperature sensors were offline (days 25-32) and due to other idiosyncrasies of interactions between the OPC server and our conduit. In particular, when our conduit re-connects with the OPC server, the protocol sends out the current values of *all* sensors irrespective of when the previous value was logged. As seen from Figure 3(c), the delta-value and the aged delta-value filtering methods still perform better than the default filtering achieving almost 77% to 98% compressions.

2) Temperature-Aware Workload Placement: Current work examines the relationship between workload placement and cooling costs [13]. Figures 4(a) and 4(b) compare the cooling costs and heat recirculation levels — the amount of heat coming from servers that returns to the inlets of other servers before returning to the air conditioning cycle — for four workload placement algorithms in a simulated 1,120-server data center: random placement (which approaches a uniform distribution over time), selecting the servers in the coldest portions of the room, and algorithms that attempt to minimize and maximize the amount of heat recirculation. Results show that cooling costs in a moderately-sized data center are significantly lower when using temperature-aware workload placement. These savings can represent tens to hundreds of thousands of dollars per year.

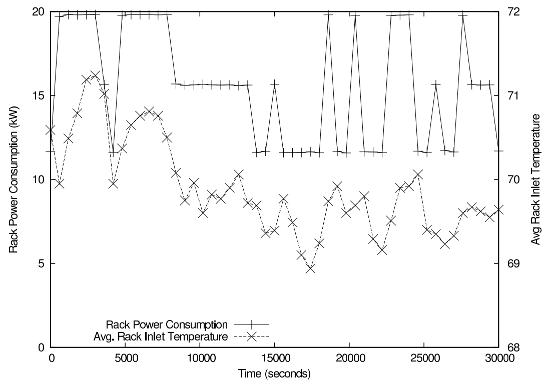
The next stage in this research is to apply these algorithms to a real-world data center. Splice is a crucial component in this step, allowing us to perform load-balancing in a way that combines temperature readings with current server utilization.

B. Workload Emulation

A workload management component that understands the detailed relationship between server utilization and temperature increase within a single server enables a fine-grained level of control over per-server power allocations. The two primary factors dictating server power draw — and, consequently, the temperature of the exhaust air — are whether or not the server is on, and the server's CPU utilization. Figure 5(a) shows the relationship between CPU utilization and temperature increase for an HP DL360 G3 with two HyperThread-capable processors, presenting four logical processors to the OS. As



(a) Using sstress to explore the relationship between target CPU utilization, power draw, and internal temperature on an HP DL360-G3. There is one sensor between the CPUs and one near the PCI (I/O) slots. Bounding bars represent a 90% confidence interval.



(b) Power draw and average inlet temperature of a rack of servers during an 8-hour experiment in the HP UDC. Sstress drove the CPU utilization changes, and Splice captured the relationship between power draw and inlet temperatures.

sstress increases target CPU utilization, OS and hardware limitations prevent it from reaching its goals; however, the server continues to draw more power. Userspace performance is not necessarily an accurate indicator of power consumption.

On a larger scale, sstress allows us to explore the relationship between CPU utilization and power at the granularity of a rack of data center server. Figure 5(b) shows the total power consumption and average inlet temperature for a rack of servers during an 8-hour experiment in the HP UDC. We see the time-delayed nature of this correlation, as it takes several minutes for the increases in server heat to propagate through the data center, and several more minutes for the ambient temperature to cool after the server return to an idle state.

V. Conclusion

As enterprise IT systems continue to respond to the changes in recent years, it becomes increasingly important for corresponding changes in methods and tools targeting system instrumentation, analysis, replay, and emulation. In particular, this paper focuses on two key needs: the need for monitoring approaches that address system instrumentation holistically across the IT and facilities domains, and the need for more detailed and flexible emulation of current complex multi-user and multi-application consolidated data center environments.

To address the first need, our work develops the notion of knowledge planes extended to include location and spatial information and environmental sensor information. As an illustration of such an extended knowledge plane, we architect the Splice framework that combines environmental sensor readings with performance measures all normalized to a common spatial frame of reference. Our proposed architecture includes a data communication and filtering engine and a database schema implemented on top of a relational database and is designed to support easy extensibility, scalability, and support for the notion of higher-level object views and events in the data center.

To address the second need, we are developing a broader data analysis and playback framework. Our data analysis captures attribute behavior trends and inter-attribute correlation properties through the use of corresponding mathematical techniques (Hidden Markov Models, EM clustering, etc) to enable us to capture and condense important statistical properties of system behavior logs. Our data playback tools seek to recreate system resource usage conditions at specified user levels of interest. As specific examples of such tools, we have built the SeASR and sstress applications and used them in conjunction with Splice. SeASR works in conjunction with Splice to generate statistical summaries that enable more aggressive filtering for scalability, detection of aberrant behavior and generation of synthetic traces for experimentation. sstress uses trace information collected from Splice to mimic resource utilization behavior at the system component level.

Though the discussions in the paper primarily focus on the implementation aspects of the tools, we have used these tools in our ongoing research on power management in enterprise environments as well as in broader resource management issues with utility computing environments. Our experiences validate the benefits from such a toolkit. For example, our recent work on temperature-aware resource provisioning [13] demonstrates that an instrumentation approach like Splice can facilitate optimizations that were otherwise not possible and reduce coolings costs in the data center by up to 40%. Similarly, our work using sstress and SeASR have been very valuable in quickly duplicating complex IT environments to understand and improve resource usage patterns [9], [7].

While our work is a good first step in this direction, we are just beginning to scratch the surface, in terms of addressing the needs that exist for such tools. The functionality of extended knowledge planes such as Splice can be significantly enhanced by the consideration of additional topological relationships that go beyond simple cooling provisioning - for example power grid provisioning, network redundancy provisioning, etc. We are also currently exploring additional techniques to strengthen the mathematical analysis in our toolkits. Similarly, workload playback tools in the context of being able to relate to higher-level application response times is an open research question. As current trends towards increased complexity in workload and datacenter configurations for enterprise environments continue, it will become even more important to address the need for correspondingly more sophistication in the methods and tools for analysis, replay, and emulation.

APPENDIX I
ACKNOWLEDGMENTS

We would like to thank Chandrakant Patel, Cullen Bash, Monem Beitelmal, and Ratnesh Sharma for their invaluable assistance with all things cooling and heat related. We would also like to thank April Slayden and Subu Iyer for their help in making the Splice query and display interface more usable than an uninteresting SQL command prompt, along with other useful modifications and suggestions for real-world deployment.

REFERENCES

[1] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, Bolton Landing, NY, Oct. 2003.

[2] J. S. Chase, L. E. Grit, D. E. Irwin, J. D. Moore, and S. E. Sprenkle. Dynamic virtual clusters in a grid site manager. In *Proceedings of the Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.

[3] M. Chen, E. Kiciman, A. Accardi, A. Fox, and E. Brewer. Using runtime paths for macro analysis. In *Proc. HotOS-IX*, Kauai, HI, May 2003.

[4] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic systems. In *Proc. 2002 Intl. Conf. on Dependable Systems and Networks*, pages 595–604, Washington, DC, June 2002.

[5] D. Clark, C. Partridge, J. C. Ramming, and J. Wroclawski. A knowledge plane for the Internet. In *Proceedings of ACM SIGCOMM*, August 2003.

[6] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC)*, August 2001.

[7] D. Economous, C. Kozyrakis, and P. Ranganathan. Modeling and understanding server power consumption. In *Hewlett Packard Technical Report*, 2004.

[8] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proceedings of 19th International Conference on Very Large Databases (VLDB)*, September 2003.

[9] D. Irwin and et al. Dense and powerless: Hardware-software coordination for blade serverpower reduction. In *Hewlett Packard Technical Report*, 2004.

[10] R. Isaacs and P. Barham. Performance analysis in loosely-coupled distributed systems. In *7th CaberNet Radicals Workshop*, Bertinoro, Italy, Oct. 2002.

[11] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation service for ad-hoc sensor networks. In *Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI)*, December 2002.

[12] J. D. Mitchell-Jackson. Energy needs in an internet economy: a closer look at data centers. Master's thesis, University of California, Berkeley, 2001.

[13] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making Scheduling "Cool": Temperature-Aware Workload Placement in Data Centers. In *2005 Usenix Technical Conference*, April 2005.

[14] C. Patel, R. Sharma, C. Bash, and A. Beitelmal. Thermal Considerations in Cooling Large Scale High Compute Density Data Centers. In *ITherm 2002 - Eighth Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, May 2002.

[15] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286, February 1989.

[16] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and Accuracy in a Large-scale Network Emulator. In *Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI)*, pages 271–284, December 2002.

[17] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.

[18] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An information plane for networked systems. In *Proceedings of ACM HotNets-II*, November 2003.

[19] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.