

A Sense of Place: Toward a Location-aware Information Plane for Data Centers

Justin Moore and Jeff Chase*
Department of Computer Science
Duke University

{justin,chase}@cs.duke.edu

Keith Farkas and Partha Ranganathan
HP Labs, Palo Alto

{keith.farkas,partha.ranganathan}@hp.com

Abstract

The continuing drive to improve operating efficiency of information technology is motivating the development of knowledge planes or frameworks for coordinated monitoring and control of large data computing infrastructures. In this paper, we propose the notion of a *location- and environment-aware extended knowledge plane*. As an illustration of such an extended knowledge plane, we architect the *Splice* framework, that extends the knowledge plane to include data from environmental sensors and the notions of physical location and spatial and topological relationships with respect to facilities-level support systems. Our proposed architecture is designed to support easy extensibility, scalability, and support the notion of higher-level object views and events in the data center. Using the above architecture, we demonstrate the richness of queries facilitated by *Splice* and discuss their potential for automating several categories of data center maintenance and control. We also discuss our experience with deploying *Splice* on real-world data centers and discuss the value from *Splice* in the context on one specific optimization that would have otherwise not been possible without the extended knowledge plane. Finally, we also provide evidence of the scalability of this deployment with number of readings, both in terms of database storage and query performance.

1 Introduction

A continuing trend is the drive to improve the operating efficiency of information technology (IT) by reducing the costs to operate the physical infrastructure. In particular, server infrastructure is increasingly consolidated in

data centers and server networks under unified administration. As the complexity and scale of these systems grows, it is increasingly hard for facilities personnel to monitor the thousands of systems, locate those that have failed, and diagnose problems in the interconnected systems for power, communications, and cooling.

This concern has motivated recent work on frameworks for coordinated monitoring and control of large-scale computing infrastructures. 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 [10] and queries on massive sensor fields [14, 8] for wide-area infrastructures such as Planet-Lab [15] or the Grid [5]. The second is to develop analysis tools to recognize patterns and diagnose anomalies in the data [3, 2, 1, 9]. 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 [4].

This paper explores a new dimension of the emerging knowledge plane: the role of environmental sensors, physical location, and spatial and topological relationships with respect to support systems such as cooling and power distribution. We claim that this information is needed in any comprehensive monitoring system, and

*This work is supported in part by the U.S. National Science Foundation (EIA-9972879) and by HP Labs. Justin Moore was supported in part by an HP Labs Internship.

that it is important to integrate it with conventional metrics such as system utilization and performance. For example, location is useful to isolate and localize problems that require manual intervention. It can also help to predict failures resulting from environmental conditions, and to improve availability by selecting resources to avoid a common scope of failures, such as those resulting from destruction of a building to failure of a shared power distribution system.

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 [11]). 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 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 [12], 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.

To illustrate the role of environmental sensors and location in an information plane for a data center, we report on experiments with an instrumentation infrastructure for a data center comprising approximately 320 systems. Our test data center is instrumented with temperature sensors and monitors for power consumption. A key element of our work is a database engine to filter sensor data and index it in an archive supporting an SQL query interface optimized for selected spatial queries. This component—called *Splice*—combines environmental sensor readings with performance measures collected from a standard instrumentation framework such as HP OpenView or Ganglia [13], and normalizes all readings to a common spatial frame of reference. *Splice* can act as a building block for a variety of analysis tools, including correlation analysis on an archived history as well as event triggers and other queries over the current state of the infrastructure.

We begin in Section 2 with an overview of the *Splice* architecture and data model and a summary of related work. Section 3 discusses the architecture and design of the key system components. Then, in Section 4, we discuss important queries enabled by a location-aware and environment-aware information plane. Section 5 presents results illustrating the correlations of workload performance metrics and environmental data, and evaluating the

performance and scalability of the *Splice* aggregation and query interface.

2 Overview and Related Work

We designed *Splice* as a component of a unified monitoring and control system for a utility data center, although it is not limited to that context. Our model includes a set of data sources or *sensors* that export current values for named metrics of interest, a set of consumers or *sinks* that import sensor readings, and a network that disseminates streams of sensor readings to sinks, such as an overlay network based on a publish/subscribe model. The sensor set includes physical sensors, e.g., for temperature or other environmental conditions that might affect equipment functioning, such as dust, humidity, or electromagnetic noise.

In this context, *Splice* is a component that subscribes to some set of sensors, e.g., all of the sensors in a data center or region. Figure 1 depicts the structure of *Splice*. Its purpose is to filter and normalize the sensor readings, integrate them with other configuration state, and archive them, indexed by time and location. It exports an SQL query interface to higher-level tools or *agents* that access the data, e.g., for analysis tasks as described below. Note that *Splice* is compatible with any method of disseminating the sensor data, and that it may coexist with other data consumers or intermediaries that process or log raw sensor streams.

Each sensor is associated with an *object* occupying a specific *location* in a three-dimensional coordinate space. Examples of objects are servers, switches, storage units, or standalone sensing devices such as the Motes used as temperature sensors in our test data center. Each object is associated with an extensible set of named *attributes* with string values, similar to LDAP or SNMP. Object attributes include a type, a location, and dynamic attributes derived from the sensors currently bound to that object. Administrative tools (agents) may define or update additional attributes for an object, e.g., to record the physical or software configuration of a server. For example, an administrative agent might populate the database with a data center configuration specified externally using Data Center Markup Language (DCML [7]), Microsoft's Systems Definition Model, or the Common Information Model defined by the Distributed Management Task Force [6].

The *Splice* data model does not distinguish between the dynamic attributes and configuration attributes normally considered to be *fixed*. The fixed attributes include data that changes rarely, such as physical configuration data, including memory size, physical relationships among components (e.g., system *X* is located in rack *R*), and properties related to location (e.g., the power circuit feed-

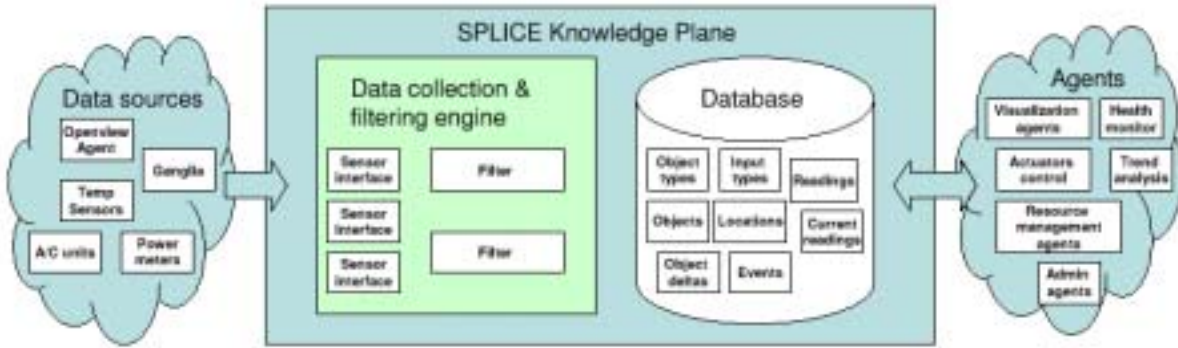


Figure 1: Splice consists of an engine for data aggregation and filtering that interfaces with a dynamic collection of data sources (sensors). It archives a filtered history of sensor readings and other object attributes in a database, and exports an SQL query interface to management and analysis tools (agents).

ing a given system). These attributes are represented and processed in the same way as dynamic sensor readings. This principle allows the system to naturally record a history of the evolution of the data center over its lifetime (see Section 3).

Agents may use the Splice query interface to perform a variety of analysis and management tasks. For example, they may monitor status in real time for health monitoring and problem determination, or to drive control policies in response to events. Agents may query the data along multiple dimensions, involving the history of object sets defined by arbitrary collections of attributes, or activity within specified regions and locations over specified time intervals. Section 4 discusses the rich set of queries made possible by integrating environmental and location data into the archived history.

Splice is built above a relational database package to handle these queries efficiently. This use of a database does not require full transactional atomicity or concurrency control: each sensor reading is an independent and immutable update, and there is no requirement for atomic update or retrieval of groups of sensor readings. Our prototype uses *mysql*, which does not impose transactional overheads.

2.1 Related Work

Splice complements other projects working on different aspects of massive instrumentation. For example, others have addressed the design of scalable overlay structures to disseminate sensor information to distributed analysis components. Astrolabe [14] is one related project that has addressed this issue. While any structured overlay could be used, Astrolabe propagates information among

zones organized to reflect a logical administrative hierarchy, similar to LDAP or DNS. One strength of Astrolabe is that attribute values are replicated, improving the robustness of the instrumentation layer.

Astrolabe supports distributed queries over attributes, interposing operators for information reduction and aggregation along the zone hierarchy. Similar approaches appear in sensor networks that construct ad hoc overlays based on proximity, and combine and process information—including spatial information—as it moves through the overlay (e.g., TAG [10]).

Other recent work has addressed massively distributed queries over current readings in sensors distributed across a wide area (e.g., Sophia [15] and PIER [8]). PIER uses an overlay to combine partial results from distributed query engines.

Relative to these efforts, Splice focuses on queries over history and uses location to integrate environmental sensors into an information/knowledge plane for compute infrastructure. Splice could function as a component in each of these systems, to extend it to the wide area or improve its robustness. For example, it is possible to deploy Splice aggregators in a hierarchy, in which each aggregator subscribes to sensor data from its local region, and acts as a query engine for hierarchical queries across many aggregators, as in PIER or Astrolabe.

There has been much recent progress on statistical analysis tools that infer component relationships from histories of interaction patterns (e.g., from packet straces) [3, 2, 1, 9]. This work has demonstrated the value of pervasive instrumentation as a basis for problem determination and performance profiling.

3 Splice Architecture

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.

This section outlines the design of Splice, focusing on the choices we made to meet these goals.

3.1 Data collection and filtering engine

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; Section 5 discusses this issue in more detail.

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.

3.2 Database schema

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 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,

Object Types <ul style="list-style-type: none"> object type id size (x,y,z) label description 	Input Types <ul style="list-style-type: none"> input type id units label description 	Readings <ul style="list-style-type: none"> time object id location id input type id value event id
Objects <ul style="list-style-type: none"> object id object type id location id parent object id is valid label description 	Locations <ul style="list-style-type: none"> location id coordinate (x, y, z) label description 	Current Readings <ul style="list-style-type: none"> time object id location id input type id value event id
Objects Deltas <ul style="list-style-type: none"> time object id location id (new, old) parent object id (new, old) event id 		Events <ul style="list-style-type: none"> time event id object id type description

Figure 2: Database schema of the Splice architecture.

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.

4 Extending the Information Plane

In this section, we illustrate the power of incorporating location-awareness and environmental data (power, temperature) into an information plane. Extending the information plane in this way facilitates data center automation for several new categories of maintenance and control. Table 1 gives some examples of the kinds of rich queries possible with Splice, broadly classified into five categories based on their usage. This list is intended to be suggestive and illustrative: some of these queries may require substantial post-processing within the analysis tools, and many useful queries are not included in this list.

The first class of queries deal with simple data collection and to monitor data center health and to present resource status to a human administrator. These queries may focus on individual sensor or system readings or collate summaries over multiple readings clustered by geographic location (e.g., servers in a rack), application grouping (systems associated with an application), or by some other form of topological grouping (e.g., systems in a power-grid).

The second class of queries extend simple collection for monitoring to include some basic intelligence to identify fault patterns or automate maintenance tasks. For example, for services that provide high availability through running multiple instances of the same service in parallel, it is important to avoid running multiple instances on systems that share a common infrastructure dependence or failure scope. Such dependencies include being located in same geographic region such as a cooling zone, room, or building, or sharing the same power circuits. Simple Splice queries can identify such dependencies ahead of time, as well as detect potential overloading of power or thermal limits under coincident heavy loads.

Other queries in this class include monitoring systems for thermal failure and characterizing the availability of systems based on their past histories. Finally, when a system fails, location information is required in order to physically find it and repair it. Though this is a relatively simple query in Splice, this addresses a significant problem in many current data centers that are densely populated or have significant churn in the location of individual servers.

The third and fourth classes of example queries listed in

Visualization and monitoring of the data center

What are the current power readings for a subset of the nodes associated with some physical meaning (e.g., power-grid)?
What is the current average temperature for all racks that are more than some distance away from another location or object?

Problem detection/avoidance and task automation

Which racks have had power or temperature rise above threshold T over time interval I , and where are they located?
Which servers have had more than D units of downtime over time interval I , and where are they located?
Which servers in a redundantly provisioned cluster share the same power circuit or are in the same fault scope?
Is the backup node for server S within 3 miles of its location?
When a node fails or needs a hardware upgrade, where is it physically located?

Understanding historical data for better data center designs and policies

What was the power and utilization of the machines in a given rack during time interval I ?
What is the average power consumption of the tier-1 workload in a three-tier system that ran on a given set of machines over a period of a week?
For a given period of time, which locations in the data center were most often hotspots?
In the last hour, how many times did the temperature at location L rise above threshold T ?
What was the difference in aggregate power consumption over time interval I across the power grids in a data center?
How many of the servers that received memory upgrades last month have failed in the last week?

Understanding causal relationships between events

How does the power dissipation impact inlet temperature at the racks?
When a cooling unit fails, what are the servers that show thermal redlining and how long does it take for the temperature to reach within a threshold of the redline limit?
What is the impact on the inlet temperature for certain predefined nodes for a given change in the resource utilization of a given rack?
Over a period of a year, is there a correlation between server downtime and the average temperature at its location?

Understanding resource properties for dynamic resource allocation decisions

What are the nodes that are present in a cool region, have a CPU load below threshold T and are located no more than one rack apart?
Which nodes have been up for longer than two months and have lifetime failure rates below threshold T ?

Table 1: Example queries enabled by Splice.

Table 1 demonstrate how Splice can be used to parse the collected data to understand historical trends and causality in events. This information can be used to optimize the data center design or improve policies for resource management. Splice can be used to understand the average behavior of a metric of interest over time either for an individual sensor or for some topological grouping of readings. For example, it could be used to understand the thermal profile of hotspots and coolspots in a data center for a given cooling configuration and relate it to system thermal redlining limits. Such averages, when computed for power, can also be used in aid of power grid balancing. Similarly, understanding correlations between server resource utilization, power consumption, inlet temperature variation, and need for cooling for a given data center can lead to significant optimizations in the design of the data center to reduce overprovisioning of power and cooling resources. Finally, causal information can also be used to determine the root cause of failure.

The final class of example queries help in understanding dynamic resource utilization properties of the data center to improve policies for workload placement and resource allocation in the data center. For example, a snapshot of system resource utilization across a rack of servers that constitute a solution (e.g., a three-tier workload) can aid in power-sensitive workload placement to improve power consumption. This technique is based on the observation that when workloads exercise systems differently it is possible to improve overall energy efficiency by allocating workloads to heterogeneous servers in a utility data center. Other similar heuristics can be envisioned that improve costs associated with cooling or risk tolerance by leveraging information about prior dynamic changes in the system from Splice.

4.1 Test Data Center

We conducted several experiments on a test data center at HP Labs (HPL). The HPL facility houses the first production installation of the Utility Data Center (UDC) solution from HP's Infrastructure Solutions division. The data center contains approximately 320 servers and over 40 TBytes of storage. This environment is set up to run the entire production load of HP Labs as well as the computing needs of the individual research groups. The room power and cooling, as well as the operational parameters of all the components are instrumented to aid the data aggregation for the Smart Data Center research program.

The HP UDC has four primary data sources.

- **Power meters:** An OLE for Process Control (OPC) server maintains the one-minute average power consumption for each rack. This data is logged in a Sybase database and exported through a DCOM net-

work service. The server updates power values once every sixty seconds.

- **Temperature sensors:** A separate OPC infrastructure monitors temperature through a series of wired sensors that are attached to the front and back of the racks. Temperature data is only available through the DCOM network interface. OPC clients can specify what temperature delta is sufficient to trigger a callback message over the network containing the new value.
- **HP OpenView:** The performance infrastructure uses Service Reporter on the server and Performance Agent on the clients. Service Reporter polls the clients periodically and updates its MS-SQL database with traditional measurements: load averages, memory use, disk utilization, and others.
- **Configuration descriptions:** Location information was fed into the database through an XML description of the room created by the system administrator.

4.2 Cooling-Aware Workload Placement

The previous section highlighted several classes of queries possible with Splice and their potential benefits. In this section, we focus on one particular example in the context of our instantiations of Splice. Specifically, we discuss how Splice could be used as a basis for cooling-aware workload placement.

As noted in Section 1, there is growing interest in improving the energy efficiency of data centers. Approximately 1/3 of the total amount of energy consumed by a data center is consumed by the cooling resources (e.g., air conditioners, cooling towers, pumps). One possibility is to allocate workloads so as to minimize the energy required to extract the heat they produce from the data center. Conventional raised-floor cooling tends to cool unevenly, thus some locations are inherently cooler than others. The data center scheduler can save energy by scheduling workload on the machines in these cool locations, minimizing the load increase on the cooling system.

Figure 4.2 illustrates this effect for a row of systems in the HP Labs data center. This figure plots the median temperature recorded by 12 temperature sensors over a 23 day period. These sensors were positioned to measure the inlet temperatures of 72 systems arranged in 6 racks¹. The figure also gives the 20th and 80th percentile temperatures, indicating the temperature spread. Observe that this spread is typically 4 deg F and increases to as much as 7 deg F.

¹While more sensors would give more temperature precision, 12 is sufficient for mapping out the thermal zones for these 72 machines.

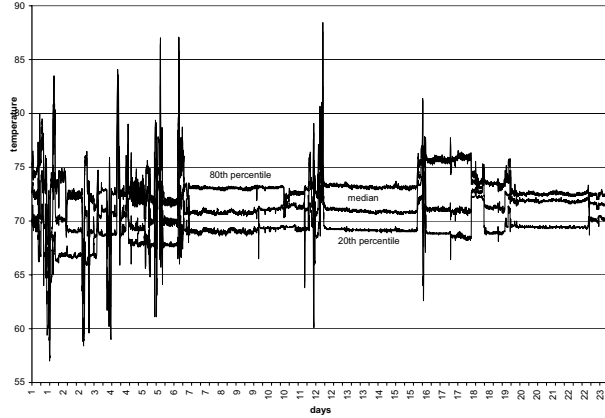


Figure 3: The median and range of inlet temperatures for 6 racks reported by 12 temperature sensors.

This spread occurs in part because the temperature of the air entering systems at the top of racks is several degrees hotter than that entering systems at the bottom. In addition, owing to complex air flow patterns, systems located in different parts of a data center can experience radically different inlet air temperatures.

Cooling-aware workload allocation requires the historical and present-time inlet temperatures for all available and suitable systems. This information is readily available from Splice, as illustrated in Figure 4.2. From this information, we can select the temperature sensors having the lowest temperature, and hence allocate the workload to the systems in the vicinity of these sensors, taking into account other relevant attributes of those systems, such as their resource types and current utilizations.

For the same sensors shown in Figure 4.2, Figure 4.2 plots the three temperature sensors with the lowest reading. Each data point (x, y) indicates that the temperature sensor y yielded one of the three lowest readings over the five-minute interval x .

From the figure we can see that certain rack positions are often but not always among the coolest. Hence, there is an energy advantage to allocating workloads to those positions, since doing so would reduce, possibly eliminate, the amount of additional cooling required. However, the location of these positions relative to other racks and objects must also be taken into account. One reason is that a high power workload at the end of a row of racks can cause the inlet temperatures of some of the machines in this rack to increase. This effect results from hot exhaust air drawn around the end of the row and into the inlets of the machines; such recirculation is much less likely to occur for machines in the middle of the row.

Figure 5 illustrates this recirculation effect. This figure shows the average inlet temperatures for each of the 6

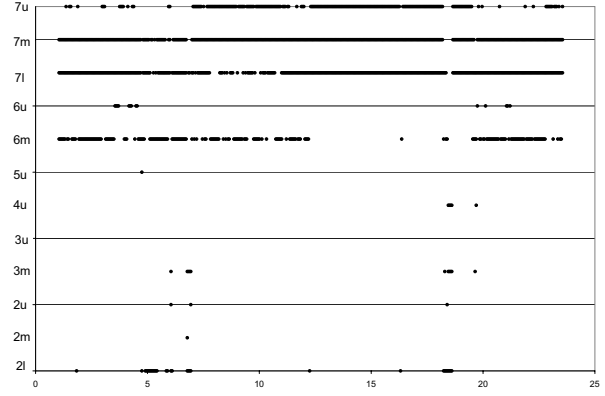


Figure 4: The three sensors with the lowest average temperature, computed every five minutes, over 23 days. The Y-axis labels provide the location of the sensors. E.g., sensor 2l is located near the bottom of rack 2, 2m near the middle, and 2u near the top.

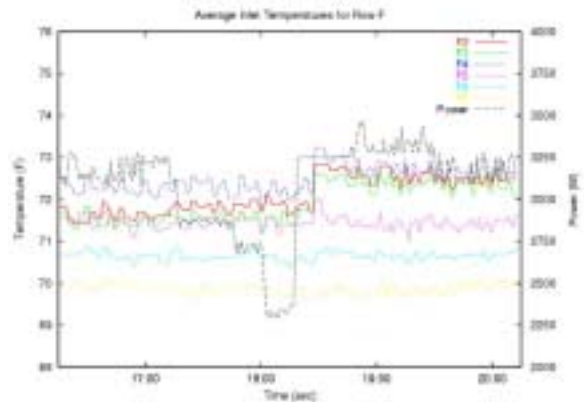


Figure 5: The inlet temperatures of the 6 racks when a high power workload was deployed at time T on 10 of the 12 machines in the rack at one end of the row.

racks described above, when a new workload is deployed on 10 of the 12 machines located in the rack at one end of the row.

Hence, an effective policy must take into account the temperature distribution in the data center, the position of the cooler spots relative to the rows and other objects, and the likely power consumption of the workload. Clearly, to implement such a policy, our extensions to the knowledge plane are required.

5 Experimental Results

In this section 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 po-

Metric	Interval (s)	Avg. Value	Avg. Delta	Dev. Delta	P(0—0)	P(!0—!0)
Power (W)	60.4	1481	0.0	95.5	0.83	0.91
Temp. (F)	153	68.6	0.0	1.06	0.37	0.81
1-min load	265	0.96	0.0	0.61	0.75	0.73

Table 2: Important Characteristics of Data Flows. Power and temperature metrics are from the HP UDC, while load average and memory utilization are taken from the Duke trace. As temperature data is partially pre-filtered on the OPC end of the conduit, the probability of three consecutive identical readings – two deltas with a value of zero – is significantly lower than for the other metrics.

tential 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 post-processor 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. We view historical data as useful for examining general trends, and for the creation and tuning of models for MAPE control loops. However, to ensure that old data does not indiscriminately get filtered out, there is an upper bound on the coarseness of the filtering granularity.

5.1 Classifying Data Flows

In order to choose a given filtering scheme and select optimal parameters, we must first understand certain characteristics of the incoming data flows. How often do the values change, and how much do they change? Is the rate of change itself constant, or are the measured values constant for extended periods of time with only bursts of activity? Table 2 shows

- the average time between readings
- the average measured value
- the average delta when the delta is non-zero
- the standard deviation of the delta
- the probability that the delta between readings r_n and r_{n+1} will be zero, given that the delta between readings r_{n-1} and r_n was zero, and
- the probability that the delta between readings r_n and r_{n+1} will be non-zero, given that the delta between readings r_{n-1} and r_n was non-zero.

Note that a delta of zero is a special case that we separate from other statistical properties. The distinction between a delta of zero and non-zero is important; if we did not make this distinction it would be impossible to distinguish between a trace that changed steadily with a relatively constant delta, and a trace that is largely quiescent but with large bursts of activity. Both could have the same average and standard deviation for their delta properties, but only the first trace is amenable to delta-value filtering.

In spite of our rule for setting aside delta values of zero while calculating data flow properties, the average delta for each measurement is zero. In calculating the average delta, we do not use the absolute value of each delta; we simply sum the differences. An average delta of zero indicates that over a long enough time window – weeks for the load average, and months for power consumption and temperature – the values are relatively stable. Assuming that were not the case, and instead the average power or temperature delta was non-zero, that would indicate that the total power consumption or temperature in the data-center would approach either zero or infinity. One might see how these are less-than-desireable (or realistic) scenarios.

The true characteristics of the data flow are its “laziness” – measured by $P(\delta_{n+1} = 0 | \delta_n = 0)$ – its burstiness – measured by $P(\delta_{n+1}! = 0 | \delta_n! = 0)$ – and the magnitude of the changes – indicated by the standard deviation of the delta.

5.2 Exploring Filtering

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

Figure 6 illustrates the effect of these filtering modes for three kinds of sensors: power, temperature, and one-minute CPU load. The CPU data (Figure 6(a)) is presented for the Duke facility while the power (Figure 6(b)) and temperature (Figure 6(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 6(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. This is due to the relatively large $P(\delta_{n+1} = 0 | \delta_n = 0)$ and $P(\delta_{n+1} \neq 0 | \delta_n \neq 0)$ attributes of these data flows. Figure 6 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 virtue of its greater compression of older values.

Figure 6(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 6(c), the delta-value and

the aged delta-value filtering methods still perform better than the default filtering achieving almost 77% to 98% compressions.

5.3 Query Times

Table 3 lists the time taken to process three representative queries that stress different axes of time, space, and objects:

- Q_1 What is the average temperature right now for all racks less than some distance away from some location?
- Q_2 Find a row with an average inlet temperature below some threshold and that has four machines with a load less than 0.75.
- Q_3 For the time interval X-Y, what racks had power consumption of above a certain threshold and where are they in the data center?

For each query (horizontal row in the table), we present data for three database sizes corresponding to a week, two weeks, and three weeks of sensor information. For each database size, we present the query speed assuming databases with (Delta-Value) and without (Raw) filtering.

As seen from the results, in the absence of filtering, query performance for archived (i.e., non-current) data is very poor. For example, the response for query 3 time goes from a five seconds to 17 minutes when we disable filtering. In contrast, as the results in Table 3 show, the query performance for the database with filtering turned on is more manageable. For example, the response time for query 3 goes from five seconds to 95 seconds when the database size increases from one week to three weeks. Deeper analysis revealed that this jump was due to the default MySQL configuration allocating small buffers to sort ordered data; unfortunately we did not have enough time to replicate the results with other standard server optimizations enabled.

6 Conclusion

“Knowledge planes” or frameworks for co-ordinated monitoring and control of such large-scale systems have recently emerged as valuable tools to address the health monitoring, problem diagnosis, and closed-loop control issues associated with IT management. However, these have all predominantly focused on conventional IT-level metrics such as system utilization, memory usage, network congestion, etc.

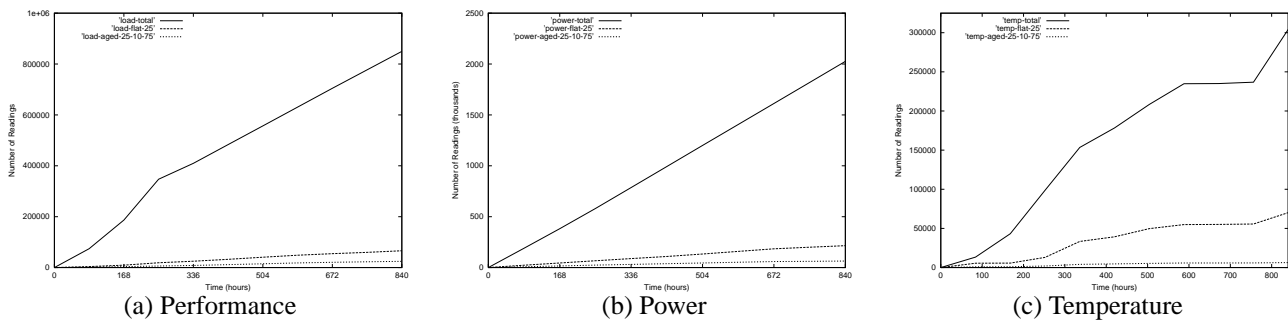


Figure 6: Demonstrating the filtering modes in Splice.

Query	1-week database		2-week database		2-week database	
	Raw	DV	Raw	DV	Raw	DV
Q_1	0.18	0.21	–	0.0	0.24	0.0
Q_2	1.18	1.25	–	0.0	1.25	0.0
Q_3	1057	5.82	–	63.37	1316	94.91

Table 3: Performance of three representative queries with and without filtering for various database sizes.

In this paper, we propose the notion of a *location- and environment-aware extended knowledge plane*. We argue that extending the knowledge plane to include data from environmental sensors and the notions of physical location and spatial and topological relationships with respect to facilities-level support systems is likely to be critical to manage and optimize future data centers. 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 the notion of higher-level object views and events in the data center.

Using the above architecture, we demonstrate the richness of queries facilitated by Splice and discuss their potential for automating several categories of data center maintenance and control. In particular, our queries illustrate how Splice can act as a building block for a variety of analysis tools include visualization agents, health monitors, resource management agents and actuator control engines. Furthermore, we discuss our experience with deploying Splice on two collections of systems, a data center at HP Labs and at Duke University, and discuss the benefits from Splice in the context on one specific agent, namely a cooling-aware workload placement engine. Our data for this experiment both motivate and discuss a potential implementation of an optimization that would have otherwise not been possible without the extended data

plane. Finally, we also demonstrate the effectiveness of our filtering methods in addressing the scalability of our solution, both in terms of database storage and query performance.

As part of ongoing work, we are examining interesting trends from topological relationships in current data centers that extend beyond the cooling subsystem that we focused on in this paper. Further, also, as we collect longer traces of data with longer passage of times, we expect the value from our Splice deployments to be even higher, particularly in identifying trends between system failure (and availability guarantees) and environmental properties of the data center.

As future data centers evolve to include ever larger number of servers operating in increasingly denser configurations, IT and facilities management in such environments is likely to emerge as one of the key challenges. We believe that knowledge planes extended to include location and spatial information and environmental sensor information, such as Splice, will be an integral part of future data centers to address these challenges.

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] 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.

- [3] 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.
- [4] D. Clark, C. Partridge, J. C. Ramming, and J. Wroclawski. A knowledge plane for the Internet. In *Proceedings of ACM SIGCOMM*, August 2003.
- [5] 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.
- [6] www.dmtf.org, 2003.
- [7] T. Howes and D. Thomas. Gaining control of complexity: The DCML standard for the data center. www.dcml.org, 2003.
- [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] R. Isaacs and P. Barham. Performance analysis in loosely-coupled distributed systems. In *7th CaberNet Radicals Workshop*, Bertinoro, Italy, Oct. 2002.
- [10] 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.
- [11] J. D. Mitchell-Jackson. Energy needs in an internet economy: a closer look at data centers. Master's thesis, University of California, Berkeley, 2001.
- [12] 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.
- [13] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler. Wide area cluster monitoring with ganglia. In *Proceedings of the IEEE Cluster 2003 Conference*, 2003.
- [14] 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.
- [15] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An information plane for networked systems. In *Proceedings of ACM HotNets-II*, November 2003.