A Systematic Approach for Improving the Quality of IT Data

Martin Arlitt, Keith Farkas, Subu Iyer, Preethi Kumaresan, Sandro Rafaeli
HP Laboratories
HPL-2008-83

Keyword(s):
data assurance, data quality, automation

Abstract:
Efforts to reduce the cost of ownership for enterprise IT environments is spurring the development and deployment of data-driven management tools. Yet, IT data is imperfect and these imperfections can lead to inappropriate decisions that have significant technical and business consequences. Current responses to imperfections in IT data are ad-hoc and incremental owing to limited awareness of the problem and to data quality being an indirect business goal. In this paper, we begin by raising awareness of the imperfect IT data problem through examples of the imperfections that occur, and a discussion of their causes and implications on IT management tasks. We then introduce a systematic approach for addressing such imperfections. Our approach allows best practices to be readily shared, simplifies the construction of IT data assurance solutions, and allows context-specific corrections to be applied during the time it takes fixes to underlying imperfection causes. We demonstrate the value of our solution through two case studies. For example, it is being used in an ongoing capacity planning effort, and reduced the (human) planner's time requirements by ≈3x to ≈6 hours, while enabling him to evaluate the data quality of ≈5x more applications and for 9 rather than 1 imperfection type.

External Posting Date: July 6, 2008 [Fulltext] Approved for External Publication

Internal Posting Date: July 6, 2008 [Fulltext]

A Systematic Approach for Improving the Quality of IT Data

Martin Arlitt
HP Labs
martin.arlitt@hp.com

Keith Farkas[†]
papers@tktk.org

Subu Iyer
HP Labs
subu.iyer@hp.com

Preethi Kumaresan[†]
UCSC
shakthi@soe.ucsc.edu

Sandro Rafaeli
HP
sandro.rafaeli@hp.com

ABSTRACT

Efforts to reduce the cost of ownership for enterprise IT environments is spurring the development and deployment of data-driven management tools. Yet, IT data is imperfect and these imperfections can lead to inappropriate decisions that have significant technical and business consequences. Current responses to imperfections in IT data are ad-hoc and incremental owing to limited awareness of the problem and to data quality being an indirect business goal. In this paper, we begin by raising awareness of the imperfect IT data problem through examples of the imperfections that occur, and a discussion of their causes and implications on IT management tasks. We then introduce a systematic approach for addressing such imperfections. Our approach allows best practices to be readily shared, simplifies the construction of IT data assurance solutions, and allows context-specific corrections to be applied during the time it takes fixes to underlying imperfection causes. We demonstrate the value of our solution through two case studies. For example, it is being used in an ongoing capacity planning effort, and reduced the (human) planner's time requirements by ≈3x to ≈6 hours, while enabling him to evaluate the data quality of ≈5x more applications and for 9 rather than 1 imperfection type.

1. INTRODUCTION

A continuing trend in enterprise computing is automating the management of the information technology (IT) infrastructure. This focus is driven by the recognition that the management of this infrastructure accounts for more than half of enterprise IT budgets [7]. Many automation tools are being introduced including tools for consolidating existing applications on to fewer servers to save recurring costs, tools for dynamically reallocating the IT resources to applications to meet business objectives, and tools for continually monitoring the infrastructure to automatically identify and diagnose problems.

Many of these solutions base their automated decisions on data, which reflects the use and behavior of the IT infrastructure. Unfortunately, this monitoring data often contains imperfections. Since poor quality data can lead to bad decisions, imperfect data can undermine the robustness and erode the reliability of any automated, data-driven management solution. Further, bad decisions distract users from their business goals, introducing overhead and additional expense. Finally, too many bad decisions will reduce the willingness of system administrators or management to (fully) automate many tasks, impacting the perceived value of the management solution.

The dynamics of current enterprise IT infrastructure complicate the task of identifying data imperfections, correcting the sources of them, and ensuring they do not reoccur. The growing scale, adaptivity, and complexity of the infrastructure and the growing volume of data makes it increasingly difficult for humans to ensure that data is properly collected and to verify its correctness in a timely manner. In addition, the growing complexity of IT infrastructures reduces the ability of humans to reason about the cause-and-effect relationships between components in the IT infrastructure, which can lead to cascading failures following an initial bad decision. Finally, searching for imperfections in data is a mundane task; human skills should be focussed elsewhere.

To address this problem, we argue there is a need for a software layer that sits between the collectors (i.e., *producers*) of data and its consumers. This *data assurance* layer is responsible for evaluating the quality of the data, correcting imperfections, and reporting quantitative data-quality measures. This layer captures best practice data assurance techniques, thus making these techniques readily available to all consumers. As such, this layer will free management tool developers from creating point solutions to individual data imperfections, an ad-hoc approach that reduces overall robustness and limits the sharing of developed solutions. The information gleaned by this layer about the imperfections in a data set can assist in identifying the root causes.

In this paper, we focus on chronologically ordered sequences of IT utilization data, that is, *IT data*. As we will demonstrate, IT data is the basis for a number of important IT management tasks. Through this focus, we raise awareness about the emerging and significant problem in enterprise computing that is arising from poor quality monitoring data. There is a common misperception that data is "perfect" or that the imperfections are harmless, and thus this important topic has not received much attention in the systems research community. In addition, we provide domain knowledge about the nature of some common imperfections

[*]This work was done while working for HP.
[†]

in IT data. Such knowledge is required as a foundation for establishing generally acceptable practices for using IT data in spite of its imperfections. Finally, we discuss our implementation of a data assurance layer for IT data. Although we focus on IT data, our solution can be extended to address other forms of monitoring data (e.g., event streams). Data consumers, data providers, and developers of IT monitoring tools and automation tools can benefit from these contributions of our work.

The remainder of the paper is organized as follows. Section 2 introduces several use cases that demonstrate the importance of IT data in enterprise environments and the need for data assurance. Section 3 addresses commonly raised questions about the need for data assurance. Section 4 discusses the types of imperfections that exist in IT data. Section 5 describes our solution for systematically addressing such imperfections while Section 6 explains the benefits of our solution through a set of (empirical) case studies. Finally, Section 7 discusses related work, and Section 8 concludes with a summary and our future directions.

2. USE CASES

To set the context for our discussion on IT data imperfections, in this section we describe several important examples of how IT data is used by enterprise management tools. IT data will play an even greater role in the future, as more management tasks are automated, thus increasing the importance of IT data quality.

2.1 Capacity Planning

The role of capacity planning is to ensure adequate IT resources are available when they are required by the business processes they support. At times, additional resources may be required to satisfy increased demands. Capacity planners seek to ensure that these additional resources are available 'just in time'; if the resources are purchased too soon, costs are incurred that could have been delayed, while if the resources are not acquired in time, revenues might be lost due to the impact on business processes.

Due to the increased emphasis on reducing the cost of enterprise IT, the importance of capacity planning is greater than ever. To improve the accuracy of capacity plans, empirical data from all servers and applications is systematically collected and stored in a database, on an on-going and regular basis. Periodically (e.g., once per month), a capacity planning tool reads the data from the database and performs a suite of analyses. In such evaluations, data imperfections (see e.g.,Section 4), can significantly perturb the analysis results, thus impacting the planning efforts. We examine this case more in Section 6.1.

2.2 Server Consolidation

A related activity to capacity planning is server consolidation. To reduce management and license costs, the workloads from underutilized systems are merged onto a smaller number of systems. The challenge is to understand how to best consolidate the workloads.

Server consolidation is often performed as a consulting service. Due to the size of most server consolidation exercises (typically hundreds or thousands of servers are considered at a time), the data collection and analysis processes are automated as much as is possible. As in the capacity planning use case, empirical IT data is collected from each server and stored in a database. Unlike the capacity planning use case, the collection phase may only last for a fixed duration of time (e.g., one month). Once the collection is complete, the consultant utilizes a tool or spreadsheet to read the data from the database and perform a set of analyses. Based on the results, the consultant recommends how to consolidate the workloads onto a smaller set of servers, and indicates whether using a small number of new, more powerful servers would be more cost effective (over the long term). Basing the recommendations on an analysis of empirical data not only enables better recommendations, but also gives the customer a customized recommendation. This case is discussed further in Section 6.2.

3. BACKGROUND

In this section, we examine several commonly asked questions about the quality of IT data and the role played by data assurance.

What is an imperfection? An imperfection is a 'blemish' in the data that loses or otherwise obfuscates details of a monitored system's behavior. Typically, numerous different imperfections coexist in a data set, which further obscures behavioral details.

Why do imperfections matter? In general, poor quality data lessens confidence in any results, observations, or conclusions drawn from that data. The specific implications, however, depend on the types of imperfections, and how the data is being used.

What causes imperfections? Two primary causes of imperfections are *complexity* and *change*. IT environments are growing in complexity with this complexity infusing the monitoring infrastructure. For example, monitoring a web portal may require agents to collect data not only about individual software components (e.g., database server throughput) but also across components (e.g., end-to-end responsiveness). This complexity increases the likelihood that the monitoring infrastructure may not be correctly set up, or that, following a change (e.g., a patch is installed), a component will no longer work as expected. Even if the monitoring tools are thoroughly tested when they are developed and installed, as the monitored IT infrastructure adapts to changes in the business needs, new errors may occur, leading to additional data imperfections.

Can't the causes just be fixed? Once an imperfection is noted, the logical next step is to fix its cause. However, in enterprise environments, it often takes a non-trivial amount of time to do this. Factors contributing to this delay include identifying the likely cause, installing patches, and rigorously testing patches before deploying them on production systems. During this time, management tools must continue to use the imperfect data. Furthermore, any change to the system (to the monitoring software or the monitored hardware or software) has the potential to introduce new imperfections into the data. Hence, addressing data imperfections must be a permanent part of any process that utilizes IT data.

Why hasn't this been solved before? Awareness of imperfections in data is not new. Yet, to the best of our knowledge, only ad-hoc solutions have been developed to date in the IT management domain. There are many pos-

sible reasons. First, historically, monitoring data has not played such a central role in the management of IT systems, and thus a systematic treatment of imperfections was not needed. Second, in enterprise IT environments, the consumers of the data (people and tools) are typically not involved in the collection and delivery of the data. Similarly, the administrators of the monitoring infrastructure are not involved in the operation of the IT infrastructure nor the development of the monitoring tools. This approach reduces the sharing of information about imperfections in the data, and requires co-ordination to address the causes. Third, the financial impact of poor data quality is often difficult to quantify. This reduces the priority to invest in proper solutions. Finally, when data quality issues are recognized as problematic, ad-hoc and incremental fixes are developed rather than a general solution to the problem. In the long term, this approach is not sustainable nor desirable.

4. IMPERFECTIONS IN IT DATA

In this section we examine the types of imperfections that arise in IT data, examine how they can be automatically identified, and list possible correction techniques. For each type, we present examples and discuss their causes. Although some of the examples may seem obvious and/or trivial (to a human), even the simplest imperfection may violate an assumption (implicit or explicit) made by the designers of the management tool that consumes the data. The result of such a violation may be incorrect interpretation of the data, unexpected behavior, or failure of a controlled system.

IT data reports how a computer system's components (e.g., CPU, memory, disk, network) are used over a period of time. Numerous different types of measurements (*metrics*) exist. Some metrics are specific to individual components, such as the utilization of each individual CPU, or of the bandwidth used on a particular network interface. IT data may also include metrics that reflect how specific processes or applications use these components. Other metrics summarize the behavior of an entire category of components, such as the average utilization of all CPUs in the system, or the aggregate I/O rate across all disks.

A useful formalism for IT data is that of a time series, which is a chronologically ordered sequence of values of a variable or set of variables (a vector) [4, 14]. For our work, we consider only *regular* time series where observations are made at equally spaced intervals [4]. This type of IT data is common. A regular time series can be thought of as a rectangular data set, where each row represents an observation, and each column represents a distinct variable.

Based on our experience, imperfections in IT data can be grouped into five generic types of imperfections; in the remainder of this section, we discuss each of these in the order they would most likely be encountered by a consumer of the data (e.g., human or tool).

4.1 Formatting Imperfections

In order to have a software tool perform data assurance tasks, it must be able to properly read and interpret the data in each observation. There are two distinct classes of data in each observation: the time stamp and the values for each metric. Numerous imperfections can be identified at this stage, from the time stamp format being inconsistent with the expected format or values having a different type than expected (e.g., an integer is expected but a string was

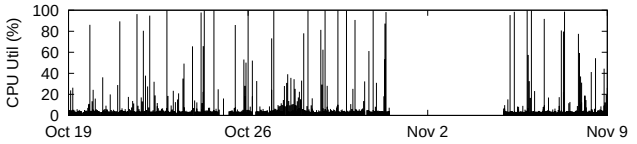

Figure 1: Missing data due to altered time format

recorded). Corrections at this initial stage may be difficult to automate, as only the most basic assumptions are tested.

When working with time series, the ability to properly interpret the time stamp for each observation is crucial. Unfortunately, many different formats are possible for expressing the same date and time. While most are easily interpreted by humans, software tools require a formal description in order to correctly extract the embedded information. For example, the HTTP/1.1 protocol specification requires compliant systems to recognize three different time stamp formats [9]. Currently, there are no widely-adopted standards to dictate which common format(s) enterprise IT monitoring tools should use. As a result, tools may not interoperate because of this problem alone.

Even within a homogeneous environment, imperfections can occur. For example, the configuration of monitoring software may be changed to provide additional functionality, or to resolve a problem. Sometimes there are unexpected side effects. Figure 1 shows a graph of CPU utilization from a single enterprise server. In this case, data is missing between November 1st and 5th. The cause of the missing data was a change in the time stamp format on the collector from 11/01/2004 (mm/dd/yyyy) to 01/11/2004 (dd/mm/yyyy). As a result, the delivery mechanism (i.e., the provider), which was statically configured to expect the first format, failed to gather the data for November 1st because it appeared to the delivery mechanism that the data the collector was offering was from January 11th.

Eventually an administrator recognized this problem, reconfigured the collector to record time stamps using the original format, and thus re-enabled the data flow to the central repository. With a data assurance layer, system administrators would be aware much sooner that a data interruption had occurred. In addition, the data assurance layer could assist the administrator in troubleshooting the cause.

4.2 Alignment Imperfections

The next task is to align each observation with its corresponding expected interval. For example, if a data set starts at midnight (00:00:00) and has an interval length of ten minutes, we would expect to see observations at 00:00:00, 00:10:00, 00:20:00, and so on. In practice, many observations captured by IT monitoring tools are slightly misaligned; e.g., 00:00:01, 00:09:59, 00:20:02, etc. These small fluctuations are due to issues like system load. In situations like this where the time stamps are 'close' to the expected interval boundary (i.e., within $\pm t$ time units), then an acceptable correction might be to rewrite the original time stamps to the expected ones. More significant misalignment can occur with IT monitoring tools that align the observations of a data set to the time at which the collection started. With such tools, it is possible to have two data sets with time stamps that do not map to the same underlying time series; for example, one data set has observations at

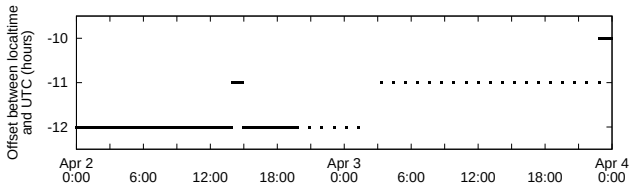

Figure 2: Incorrect and fluctuating UTC values

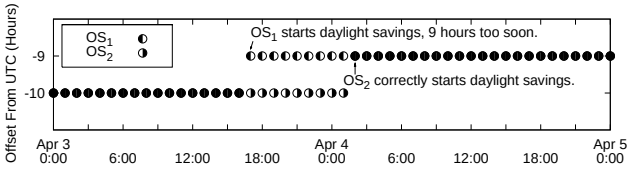

Figure 3: Incorrect daylight savings change (each dot represents the offset at the start of an hour)

00:00:00, 00:10:00, ... while the other has observations at 00:02:00, 00:12:00, ... This variation is particularly problematic when data sets from different systems are used together. When time stamps are moderately misaligned, such as in this example, alternative correction techniques, such as linear interpolation, might be applied. If the misalignment is significant, as a last resort, the observation could be discarded and treated as missing.

Another challenge for aligning observations to their corresponding expected intervals is adjusting for daylight savings. Since computer systems run continuously, time stamps should ideally be recorded in UTC (Coordinated Universal Time). Many monitoring tools, however, record time stamps in local time, which for many locations is adjusted for daylight savings twice annually. Time zone conversions are also problematic as time zone strings (e.g., GMT, UTC, PST8PDT) are not correctly interpreted by all operating systems. Consequently, a monitoring tool running on top of two different operating systems may behave differently.

Aligning observations to UTC can eliminate occurrences of missing or extra observations, as well as facilitate comparisons between different systems.[1] Unfortunately, any tool that manipulates the data can introduce imperfections that were not in the "raw" data collected by the agents. Figure 2 shows two alignment imperfections introduced by the delivery mechanism.[2]

This mechanism archived data from a number of servers in the Central time zone, then generated a UTC time stamp for each observation. Prior to the switch to daylight savings, there should have been a six hour offset between local time on the server and UTC. As Figure 2 shows, an offset of 12 hours was incorrectly calculated. In addition, Figure 2 shows four changes in the offset around the time that daylight savings started: 12 to 11 hours, 11 to 12 hours, 12 to 11 hours, and 11 to 10 hours. There should have been only a single change (from six to five hours).

Even when the conversion algorithm is correctly implemented, the results of the conversion can be imperfect. This can happen, for example, when the algorithm depends on externally provided information about the time zone, its offset from UTC, and when daylight savings starts and ends (if daylight savings is observed). As mentioned, one problem is time zone strings that are not 'portable' across operating systems. A second example is provided by Figure 3. Figure 3 shows the results of a conversion algorithm for the Hawaiian time zone, run on two different operating systems. On the first operating system (OS_1), the time zone rules provided incorrect information, and the algorithm adjusted for daylight savings nine hours too soon. OS_2 provided the correct rules, so the offset was adjusted at the appropriate time. From this example it is possible to see how imperfections can be re-introduced into an environment. Even if the time zone rules on OS_1 were corrected to fix this particular imperfection, a similar imperfection could be introduced the next time the rules are updated (e.g., the United States recently decided to extend the length of daylight savings).

In both of these cases, the inconsistencies can impact a decision reached by a management tool that correlates information gathered on different systems or that gathered on a single system when daylight savings came into effect. These inconsistencies, however, can be easily detected by a data assurance layer if the layer knows when daylight savings began, the time zone in which the data was recorded, and the expected UTC offset.

4.3 Missing Observations

Once the observations in the data set have been aligned, the next task is to identify the missing observations. This process requires meta-data about the time series; namely the start and end times, and the interval length. Using this, it is straightforward to step through the data set and determine which, if any, observations are missing.

While identifying missing observations in regular time series is straightforward, determining the proper corrective action to take is not. The first step is to narrow down on the causes of any missing data. To do this, additional meta-data is needed. For example, if the system was not running, or if the monitoring agent was disabled, no data will be available. Similarly, if the time stamps are based on local time, and the monitored system is in a locality that observes daylight savings, then there will be $\approx \frac{1\,hour}{interval\,length}$ missing observations every spring. A key aspect of our work is therefore to understand what meta-data is required, how can it be obtained or inferred, and how should it be represented so as to be useful to a data assurance layer.

To apply the best correction technique, it is important to understand the cause of missing data. For example, if the monitoring agent was disabled (e.g., to update its configuration), and the system was believed to be operating normally, then the data could be assumed to be Missing At Random (MAR) [14]. In this situation, a statistical technique such as Expectation Maximization (EM) could be used to fill in the data [14]. Depending on how many (consecutive) observations are missing, different statistical techniques might be applicable [11]. For example, if only a few observations are missing, a simple technique like linear regression or imputation [1] might be sufficient; for larger numbers of missing observations, more sophisticated techniques, like EM, could be invoked. In other situations, the observations are Not Missing At Random (NMAR), in which case other filling techniques, such as direct modeling, are more appropriate [5]. For example, if the system was offline, one could impute

[1]Time stamps must also be synchronized across systems.

[2]The dotted line in Figure 2 indicates a third imperfection in the data (missing obvservations), that were also injected by the delivery mechanism.

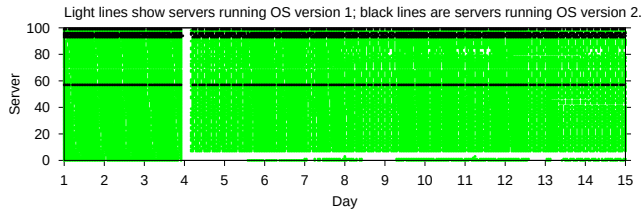

Figure 4: Missing data due to multiple causes

(i.e., fill in) zero values for metrics such as CPU utilization or network bandwidth, and use the last known value (i.e., *last observation carried forward* [8]) for metrics such as file system capacity. Section 5.1.2 discusses statistical modeling in more detail.

There are many reasons why observations may be missing from a data set. In some cases, the observations are never recorded by the agent on the monitored system. For example, the agent may rely on the operating system kernel to generate event traces of activity on the monitored system. To minimize overhead, these events may only be retained (i.e., buffered) for a short period of time (e.g., seconds or minutes) [17]. If the agent does not use the events during this period, lost observations (or inaccurate values) could result. Similarly, the delivery mechanism may fail to retrieve the data recorded by the agent (before the agent deletes it due to space constraints), thus making the data unavailable to the consumer.

It is important to realize that not only are there many causes of missing data, but that multiple causes are likely to be present when examining non-trivial data sets. This situation is illustrated in Figure 4 where the shaded region shows the availability of IT data for 100 servers over a two week period. As can be seen, there are a number of instances of missing data (indicated by white space in Figure 4), which would be problematic if the data set were used for any of the use cases discussed Section 2.

There are several causes for the missing data. First, in the early hours of day 4, no data is available for any of the servers. The most likely cause of this gap in the data is a failure with the delivery mechanism. Second, servers 3-7 are missing observations from day 4 through day 14. It appears that delivery was not re-established for these servers. The black regions of the graph (e.g., servers 57, 93-96, 99 and 100) indicate another reason why observations may appear to be missing. These servers used a different version of the operating system than the majority of servers in this data set; some of the metric names differ between the two versions of the OS. If the data consumer was not aware of this difference, they would have had no observations for this group of servers. Clearly, any decision reached using this data set must take into account its quality. The data assurance layer we propose would provide this assessment and, when feasible, correct for the missing data.

4.4 Extra Observations

Extra observations can be detected in much the same way as missing ones. Once extra observations have been discovered, it is important to distinguish between duplicate and multiple observations. A duplicate observation has identical values in all columns (i.e., for the time stamp and all metrics). Multiple observations for an interval have identical time stamps, but different values for one or more of the recorded metrics.

Correcting for duplicate observations is simple; any of the observations can be retained, while the others are discarded. Correcting for multiple observations also requires meta-data. For example, if the observations were not aligned to UTC, systems in locations that observe daylight savings will see $\approx \frac{1\,hour}{interval\,length}$ intervals with multiple observations. In this situation the proper correction is to align the observations to UTC, in which case all of the multiple observations will be retained. If the cause is not known, the observations could be treated as duplicates if the values between the observations do not differ significantly, or the observations could be dropped, and treated as missing. Which of these options is used depends on how the data is being used.

The most common cause of extra observations we have observed is abnormal behavior in the monitoring system. Perhaps the most significant concern with duplicate data is the overhead it adds to the data store at the central repository. For example, we have observed multiple instances of a delivery mechanism generating extra observations, as many as 6,000 duplicates for a single observation. These duplicates may also bias the results of certain data uses, if each duplicate observation is treated as a unique one. The occurrences of multiple observations we have observed were also due to unexpected behavior with the delivery mechanism.

4.5 Incorrect Values

The final type of imperfections we discuss are incorrect values for the observed metrics. Some recorded values are obviously incorrect; for example, values that exceed the minimum or maximum allowable value for a metric. In other cases, we may have reason to doubt the correctness of one or more values. For example, relationships may exist between variables (the total CPU utilization metric is the sum of CPU utilization metrics of all applications running in an observation); if the relationship does not hold for the values in a specific observation, we may not know which value is incorrect, but we can safely assume that at least one of them is inaccurate. Finally, there may be values that appear abnormal but there is little substantiative evidence. Statistical models can play a role in these cases.

When evidence exists that indicates a specific value or set of values is incorrect, the value(s) can be discarded and corrected using an applicable technique (as discussed under missing observations). With "uncertain" values, the context in which the data is to be used must be considered. For example, in a capacity planning exercise where it is preferable to be more conservative, uncertain values may be retained but their presence noted for later investigation.

There are many obvious imperfections in IT data, such as CPU utilization exceeding 100% or network bandwidths exceeding the capacity of the interface. Many of these imperfections are introduced by the agents, or the mechanisms on which they depend, such as kernel-generated event traces [17]. Figure 5 provides an example where the individual values were not obviously wrong, but there was a problem with the data. In this graph, we evaluate the relationship between the system-wide CPU time used and the CPU time reported for all of the applications running on the system. We would expect the residual for each observation period $r = systemwideCPUtime - \sum_{i=1}^{n} application_i CPUtime$

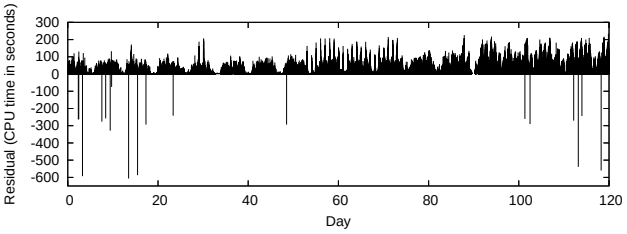

Figure 5: Relationship between metrics not upheld

to be close to zero, if all values are correct, and values are available for all applications.

Figure 5 shows a system for which there are two significant imperfections in the data.[3] First, the large positive residuals (and the periodic trend they reveal) indicate that at least one application is not being recorded (properly). Second, the large negative residuals indicate that in numerous intervals, the data reports that the applications used significantly more CPU time than the system-wide metric says was used. The first of these imperfections is caused by a configuration error; the latter by an error in the data collected by the agent on the system, or by an underlying mechanism. Both of these imperfections could have significant implications, depending on how the data is to be used.

For example, if a capacity planner failed to account for applications running on a system (as indicated by the positive residuals), the plan could oversubscribe resources on the system, and result in service level violations. With the system from Figure 5, up to 78% of the cpu time was unaccounted for (233 of 300 seconds in the interval), making oversubscription a distinct possibility. A capacity plan that incorrectly considered erroneous values (i.e., the negative residuals) would likely undersubscribe resources on the system, which could result in an additional system being purchased prematurely. For example, Figure 5 shows that on three occasions, the applications running on the system reportedly consumed more than twice the available cpu time (indicated by a residual of -600 seconds).

4.6 Summary

The five categories described in this section are sufficiently general to capture all of the imperfections we have encountered in practice for regular time series. We are aware, however, that other categories may be required as new imperfections are discovered and classified, or as other types of data (e.g., event logs) are considered. In addition, it may be beneficial to consider other taxonomies. Although it may be possible to correct for the occurrence of (some) imperfections, we believe this should only be done as a temporary solution, until the underlying cause can be fixed.

5. AN IT DATA ASSURANCE SOLUTION

In this section we present our data assurance solution for IT data. Section 5.1 discusses our approach, and Section 5.2 describes our implementation. To make the problem more tractable, our work to-date has focused on supporting data consumers that perform offline analysis of time series-based IT data. As we have already demonstrated and will reaffirm in the remainder of the paper, many important functions for managing enterprise IT environments fall into this category. We plan to address other data requirements (e.g., real-time, event-based, etc) as part of future work.

[3]The interval length in this data set was 300 seconds.

5.1 Approach

Our solution has been influenced by work we have done with developers and users of management tools for enterprise IT environments, helping them prepare data for use by these tools, understand the scope and significance of imperfections in the data, and uncover the source of the imperfections. This experience has taught us that data consumers require the following properties of a data assurance solution:

- **simple** so as to gain the trust of users and not introduce other possible failure points;
- **repeatable** so that the same imperfections are identified in the same data set each time it is processed, and similar sets of imperfections are identified in similar data sets;
- **scalable** to accomodate large volumes of data;
- **flexible** to support the different needs of data consumers, and to allow expert users to fully customize its behavior, while being *transparent* to other users;
- **extensible** to support adding new functionality.

A data assurance solution must also provide value to the organization that develops it and maintains it. Any functionality developed for a given data consumer must be readily reapplied to other data consumers without significant re-engineering. Hence the solution components must be reusable and configurable. Similarly, deploying the solution for a given data consumer should involve little or no re-engineering of the existing functionality. Thus, the solution components must also be readily composable.

In the remainder of Section 5 we describe how our solution provides the noted properties. We begin by examining our approach for detecting imperfections, correcting for them, and assessing data quality.

5.1.1 Detecting Imperfections

To detect imperfections, we use a rule-based approach. We chose this approach because it is intuitive and hence easy to communicate, and because it is exact – there is no ambiguity in whether a particular condition will violate a rule or not. The rules test for violations of the properties of the monitored environment and monitoring systems. For example, one rule checks if the value of a metric (e.g., CPU utilization) is within the range (e.g., 0-100%) defined by the corresponding data collector. A second checks if the value of a metric (e.g., memory usage) is inconsistent with the monitored environment from which the data was gathered (e.g., an application reportedly used 300 MB of memory but the system only has 256 MB). In both rules, meta-data is required; Section 5.2 discusses how this meta-data is obtained and how it is represented. Table 1 provides additional examples along with possible correction techniques (discussed in the next section). When available, additional information is also taken into account. For example, if a system was known to be offline for a period of time, missing values in this period would be noted as being expected and appropriate (e.g., null) values will be generated.

Imperfection type	Test	Required meta data	Example corrections
Formatting	time stamp or metric value is valid?	expected data types	interpolate row
Formatting	correct number of metric values?	expected number of metrics	reject row
Alignment	time stamp aligned to expected time series?	expected begin time and time step	interpolation or rewriting
Alignment	correct time zone conversions?	source time zone, UTC offset, daylight saving time	reject data row
Missing observations	irregular time stamps? missing values?	expected time series properties, number of metrics	imputation or interpolation
Incorrect values	value outside expected range for metric?	set of acceptable values for metric	set values to null
Incorrect values	$\sum_{i=1}^{n} Mapplication_i - Msystem \approx 0$?	relationships between metrics M	interpolate values

Table 1: Examples of imperfection detection rules and applicable corrections.

Our rules are those that a skilled analyst would apply when manually inspecting the data; the value of our approach is that any data consumer (human or tool) can systematically apply these rules across large volumes of data while customizing corrections and applying data enrichments (discussed in the next two sections). Thus, the expertise of a small set of analysts can be leveraged by a much larger number of consumers, in a cost effective manner. Section 6 reinforces these benefits.

We identified our rules by examining a number of data sets, frequently after a data consumer had complained about suspicious data. As this approach is time consuming and reactive rather than predictive, we are exploring extending our solution to include statistical techniques for identifying potential imperfections. One such technique is using statistical models to detect probabilistic outliers. A key challenge is identifying those techniques that are applicable to IT enterprise data, and which exhibit good repeatability and are not very complex.

5.1.2 Correcting Imperfections

To correct imperfections we use a number of approaches, many of which were requested by the developers of management tools with whom we have worked. We currently offer three basic approaches: deleting a data row or a data value, regenerating a row or a data value, or rewriting a time stamp. Table 1 illustrates some example mappings between these approaches and the imperfection-detection rules shown in the table. To regenerate a row or a data value, we use imputation (e.g., replace missing values of a metric with the mode of the metric's observed values, or set incorrect values to zero) and linear interpolation. While our rule-based approach enables us to be exact in our detection of (known types of) imperfections, corrective techniques are more subjective. In general, it is not possible to turn imperfect data into perfect data. We expect data consumers to use these techniques in conjunction with our data quality measures and individual data point meta-data (see Section 5.1.3) to make an informed decision about how to use the data.

As an initial attempt to identify more accurate corrective techniques, we investigated several statistical techniques for modeling IT data. To illustrate how such models may be used, we consider briefly the problem of filling in missing values. A number of models are commonly used for analyzing time series, including the Auto Regressive model, Moving Average model, and Auto Regressive Moving Average model [6]. While these models are desirable owing to their relative simplicity, they are not suitable for modeling non-stationary time series [6]. Unfortunately, metrics in IT data such as CPU utilization typically exhibit non-stationary behavior. For such metrics, more sophisticated such as the Generalized Auto Regressive Conditional Heteroskedasticity (GARCH) model [3] are required.

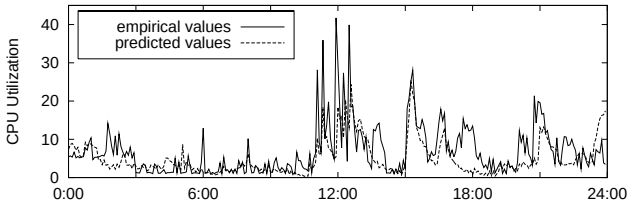

Figure 6: Missing-value prediction (GARCH model).

Figure 6 provides an example of the predictive ability of the GARCH model applied to missing CPU utilization data. In this example, a 22 week-long trace of CPU utilization data from an SAP system was used. The CPU utilization values of the fourth Sunday were removed from the trace. The GARCH model was trained on the remaining data and the parameters of the model were identified. Once the model was built, it was used to predict the CPU utilization values for the fourth Sunday. Figure 6 plots the predicted values along with the actual values. Visually, the predicted values follow the actual values that were removed from the trace reasonably well, and are obviously better than simple imputation or linear interpolation, for a gap of this duration.

While the apparent agreement in this case suggests the model is promising, a number of challenges exist for applying it in practice in a data assurance solution. First, the model has seven parameters which must be estimated. In our experiments, we found the choice of parameter values noticeably impacted the model's predictions. Hence, given that the use of such a model in a data assurance solution would require automatic estimation of the parameters, it remains unclear whether the model would meet our repeatability design objective. A second challenge is that such a model demands strong periodic trends in the data, but such trends are not always present. For these reasons, we have not yet employed any (significant) goodness of fit tests. Finally, relatively complex models appear to be required to model IT data, thus challenging our simplicity design objective. However, if a statistical technique is deemed useful for a task in this domain, it can be implemented and utilized as part of our data assurance solution.

In summary, we have examined a number of statistical techniques for the purpose of corrective data-generation. As

expected, we found that improvements are possible over the simple techniques that we currently use, but that they come at a cost. While there are undoubtedly other statistical techniques that could be better suited for this role, we argue that the correction of data should never be used in place of permanent resolution of the sources of imperfections.

5.1.3 Assessing Data Quality

Our goal in assessing data quality is to provide the consumer with information they can use to make an informed decision on whether or not to use parts or all of a data set. When the consumer is a management tool, this means providing quantifiable measures of the data quality.

Our solution provides data consumers with two mechanisms for assessing data quality. First, it provides measures that quantify the occurrences of imperfections. To assess the overall goodness of a data set, we currently use the metric $q = 1 - \frac{number\ of\ observations\ with\ imperfections}{expected\ number\ of\ observations}$, where values close to 1 indicate "high" quality data, and values significantly less than 1 indicate data of much poorer quality. This metric provides data consumers with a basis for deciding whether to use the data. Secondary measures are provided to assist in this process. These quantify the occurrence of specific imperfections globally as well for individual metrics, such as the number of missing data values and the number of inconsistent metric values.

The second mechanism provided by our data assurance solution is notations that are attached to the data values in a data set that violated a rule. These indicate the rule that was violated and the corrective action taken, information that a data consumer can use to assess on a value-by-value basis whether a data point should be considered.

5.2 Implementation

There are multiple ways a data assurance solution could be implemented, including incorporating the necessary logic into each management tool or incorporating the domain knowledge into a set of libraries that are leveraged by each tool. However, these approaches require modifying existing management tools and complicate incorporating new data assurance features. Hence, our preferred solution is a distinct tool that can be readily used by existing and new management tools. The data assurance tool applies the approach discussed in Section 5.1 to *cleanse* the "raw" data and annotate it with data quality measures and notations. This cleansed data is then used as input to the consumer's normal tool chain.

Figure 7 provides a conceptual view of our data assurance tool. The tool sits logically between a data provider and a data consumer. It comprises a set of optional translators and a number of configurable modules. For each unique type of data source, a translator is written to convert the data into a standard format that uses a specially-designed trace language to capture the meta-data. The translator also provides meta-data about the metrics provided by the data source, such as data types and defined ranges for values.

The tool supports two classes of modules: data assurance modules and data enrichment modules. While our focus is on data assurance, data consumers often require a set of common data preprocessing functions (e.g., deriving one metric from another one), and such functions are natural counterparts to data assurance functions. Each module reads the data and meta-data that is captured in the standard format, applies its specific cleansing or enriching functions to it, and writes the data and corresponding meta-data out, again, using the standard format. For those consumers that are unable to read the standard format, a second translator is used to transform the cleansed data from the standard format to the format required by the consumer. In this situation the meta-data about the cleansed data must be shared with the consumer via an alternative method, such as a separate file or database table.

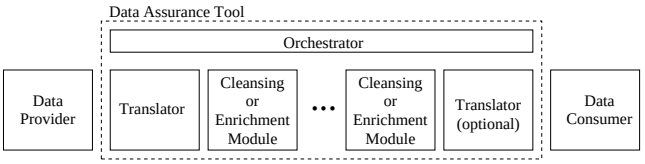

Figure 7: Conceptual view of data assurance tool

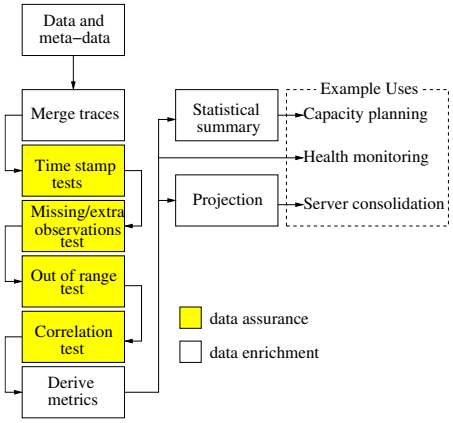

Figure 8: Conceptual view of a workflow template

Our current solution includes modules that implement the rules described in Table 1, the corrections and quality measures described in Sections 5.1.2 and 5.1.3. It also includes a module that merges related data sets, a function that is often required for rules that consider relationships between metrics. Two other enrichment modules derive new metrics from existing metrics: one computes statistical summaries while the second evaluates an algebraic expression whose terms are constants or other metrics. Finally, another module projects a data set forward in time, a function that can be used in capacity planning and server consolidation.

The order in which the modules are called and their configuration parameters are defined in a workflow description, which is implemented by the orchestrator (see Figure 7). We have created a small number of standard workflow templates, which are easily customized to generate a workflow for each data consumer. Figure 8 presents a conceptual view of a multi-purpose workflow template that can be used for a number of uses including capacity planning (requires summary statistics), health monitoring (requires cleansed data), and server consolidation (requires projected data).

As with the meta-data about a data set, the configuration parameters of each module in a workflow are described using our trace language. Hence, the trace language is used to describe such information as the format of each observation (e.g., time stamp format, number and type of metrics), the parameters of the expected time series (e.g., start time, time

```
# meta-data describing metrics present in the input trace,
# namely metric name, data type, and context for metric
metric-defn, DATETIME, type=DAY-TIME
metric-defn, CPU-UTIL, type=float, system=foobar.com

# statements commanding the cleansing module to
# perform a range check on CPU-UTIL and set values
# not in range to zero; also check generate missing values
output, DATETIME, input, isTimeBase,missing:interpolate
output, CPU-UTIL, check:0:100:set-to-zero,missing:set-to-zero

# meta data describing the input trace
data-row-separator, character=','
fields, DATETIME, CPU-UTIL
trace, type=regular-time-series, start-time=.., end-time=.., ...

# data rows fed into check module
2005-02-01 12:05:00, 15.61
2005-02-01 12:10:00, 100.66
2005-02-01 12:20:00, 10.66

# corresponding output from cleansing module, which includes
# meta-data for each metric value (see mFlag columns)
fields, DATETIME, mFlag, CPU-UTIL, mFlag
trace-type, type=regular-time-series, start-time=.., end-time=.., ...
2005-02-01 12:05:00,, 15.61 ,
# over range CPU value (vL) flagged & set to zero (stz)
2005-02-01 12:10:00, 0, vL:stz
# row was missing (vM), values generated via interpolation
# (int) or set to zero (stz) and flagged
2005-02-01 12:15:00, vM:int, 0.0, vM:stz
2005-02-01 12:20:00,, 10.66,
```

Figure 9: Trace language examples

zone, interval length), the parameters of the input data set (e.g., start time, end time, time zone), the expected ranges for the metrics of interest, the relationships between metrics, and the types of tests to be applied to each metric in a data set. In addition, the trace language describes meta-data about individual data points, such as whether the data point is null, whether it failed an imperfection test, and if so, what corrective action was taken. Figure 9 gives some example of trace language statements.

Data quality measures are computed by the individual modules as they execute a workflow and by the orchestrator using the imperfection records each module produces as it executes. These measures are provided to the consumer along with the data.

5.3 Summary

Our data assurance tool captures experience-based properties required by data consumers and tool developers. It is conceptually simple and provides repeatable results owing to the use of a rule-based approach and intuitive corrective techniques. It is scalable in that it can handle the large volumes of IT data required by management tools in a much shorter period of time than could any person.[4] It is flexible and extensible, as new modules can be quickly developed or extended. Since IT data is required in an ongoing manner, the small development and maintenance costs can quickly be recovered. In the next section, we present several use cases that put these benefits in a business context.

6. CASE STUDIES

[4]Although we currently use a single instance of the tool to cleanse all N observations in a data set, the nature of many IT data sets allows for parallelized cleansing, should further scaling of the data assurance process be required.

Table 2: Impact of invalid values on capacity plan.

Date& Time	CPU Utilization (%)	CPUs required
Sep 2, 11:30am	100.18	16.03
Sep 2, 9:30am	100.09	16.01
Sep 24, 9:50am	100.05	16.01
Sep 21, 5:10am	99.86	15.98
Sep 30, 7:30am	99.40	15.90
Sep 11, 4:00pm	1.59	0.25

In this section we describe how the generality of our data assurance solution is useful in conjunction with enterprise management tools. We provide detailed examples around capacity planning and server consolidation.

6.1 Capacity Planning

As noted in Section 2.1, capacity planning is an on-going management task in enterprise computing environments. In the recent past, we worked with a planner who is responsible for overseeing resource usage of applications running in a Shared Application Server Utility (SASU). In this utility, capacity planning is done each month using data collected every 5 minutes reflecting each system's and application's CPU and memory utilization. From the data, maximum, average, and 90th percentile values are computed for each application and system. Reports are then generated that detail such factors as the CPU and memory usage of each application, the unallocated capacity of each system, and the projected growth in resource needs of each application. Using these reports, the capacity planner determines the new resource entitlements for each application, on which system to deploy new applications, and whether the utility is running out of capacity.

Among the characteristics used in the capacity planning decision was the maximum CPU utilization of each application. When our collaboration began, the planner was aware of a type of imperfection that had a significant and detrimental effect on the capacity planning results. On occasion, these values were significantly greater than the typical values. Obviously, if these large values reflected the actual application behavior they needed to be included in the planning decisions. However, if these values were erroneous, they should not be considered, as it would (at best) result in the capacity planning algorithm dedicating too many resources to the application, resulting in an under-utilized system, and (at worst) would trigger the premature purchase of an additional system.

Table 2 provides empirical data points to illustrate the impact of erroneous values on the capacity plan for CPU assignments. Table 2 shows the six largest CPU utilization values (out of 8,640) for the month of September for one of the managed applications. The top five values all suggested the application required a full 16 CPU server. However, if these five values were ignored, the application only required one quarter of a CPU.

As a result of this type of behavior, the planner was spending ≈ 20 hours per month to manually examine the IT data for each of the 4 production systems and 75 applications. Neither the developers nor the providers had processes for systematically detecting or removing this problem.

Figure 10 shows the CPU time used over a four month

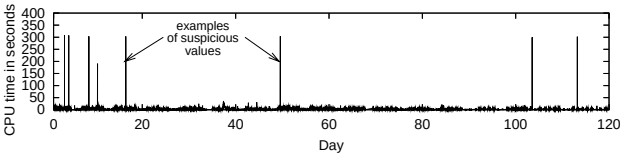

Figure 10: CPU time data for one application

period by one of the SASU applications.[5] In this figure, there are nine intervals where the CPU time value appears to be erroneous. Unfortunately, this figure does not shed any light on why these values occur. However, when we apply the negative residual test (Figure 5 shows the results of this test for all applications in this data set), we find that all nine of the suspicious values overlap with an interval that there was a large negative residual. In other words, the nine suspicious values are all incorrect, the result of a collection error. If these erroneous values are ignored, the maximum CPU time required for this application drops from 308 to 34 seconds per interval,[6] a reduction of almost 90%.

In addition to automating the discovery and correction of this type of imperfection, our data assurance solution provides a much more scalable and cost effective method for checking the IT data. Since our initial involvement, SASU has expanded to support 395 production applications running on 30 large multi-processor systems. This results in more than 7,000,000 metric values each month, a number that cannot be manually processed in a timely and cost effective manner.

Our involvement with the SASU project demonstrated several other benefits of our solution. First, using only a remote JDBC connection to the provider's data repository, we were able to examine the SASU data, and identified eight other types of imperfections. Second, we were able to implement the specific corrections the planner requested for each type of imperfection that had a significant impact on his work. We implemented corrections for all but the "positive residual" imperfection, although we do alert the planner to it. The corrected data is written to a separate table in SASU's database; the planner simply uses the data from this new table in his planning. Third, we enabled an "expert" in data imperfections to provide a customized solution to the planner, without requiring the planner to understand how to implement a module. At the same time, the expert was able to quickly create this customized solution, as existing modules could be reused.

Presently, the planner spends $\approx$ 6 hours per month on data quality issues, mostly reviewing the list of imperfections identified by our solution. This represents a time savings of $\approx$ 3x, with the planner now responsible for $\approx$ 5x more applications. Furthermore, through improved summarization of the detected observations, the planner will be able to support more applications in less time in the future. In one three month period our solution identified and corrected 16,181 imperfection observations, 14,876 of which were missing, 872 were extra, and 411 with inconsistent metric values (297 of these were identified with the negative residual test). Perhaps the best indication of our value though, is that our data assurance solution is now an integral part of the SASU capacity planning process.

We have also used our data assurance solution in several other capacity planning exercises. In these cases, IT data comes from different providers. With our modular approach, all that is required to couple these sources to our solution is the creation of a translator. Since capacity planning exercises are typically ongoing, the short-term cost of implementing the translator (or modifying an existing translation module) is far outweighed by the long-term value obtained by a generalized data assurance solution for enterprise management tools.

6.2 Server Consolidation

As discussed in Section 2.2, server consolidation exercises are often performed by consulting firms. These firms are faced with several challenges when participating in these exercises. First, they need their recommendations to be as accurate as possible. If the recommendation is too aggressive, the consolidation may be unsuccessful, resulting in problems such as dissatisfied customers and damage to the firm's reputation. If the recommendation is (overly) conservative, the firm may lose the project to a competitor. Second, consulting firms rely on different consultants to perform the studies and create the recommendations. To ensure consist recommendations, best practices must be adhered to. Data quality can obviously impact both of these; as a result, a data assurance solution helps.

Although we have not used our data assurance solution in an actual server consolidation exercise, we do have two (empirical) data sets that were collected for such purposes. In this section we use these data sets to demonstrate how a data assurance solution could enable a consulting firm to address the challenges described above.

For this case study, we consider a hypothetical consulting firm that provides server consolidation recommendations to enterprise customers. Since the firm is aware that poor quality data will lead to poor(er) recommendations, the firm institutes the following data quality policies:

- *duration:* at least one week of IT data (overlapping in time with other servers) must be available for a server to be included in the study.
- *frequency:* observations must be recorded once per hour for each server.
- *coverage:* at least 90% of the expected observations must be available to include a server in the study.

In other words, for a server to be included in the exercise, observations must be available once per hour per server for the same one week period, with no more than 10% of the expected observations missing.[7]

Figure 4 shows the first set of servers to be considered for consolidation. In this case, the first two policies hold for all 100 servers. However, 10 servers fail to satisfy the third constraint; servers 1-7 are missing $\approx$ 75% of the expected observations, while servers 48, 70 and 80 are missing slightly more than 10%. In this case, the consultant could exclude

[5] $CPU\ utilization = \frac{CPU\ time}{interval\ length}$.

[6] Only 300 seconds is available per interval for all applications on the system.

[7] This set of policies is simple by design. An actual set would require policies about a range of other constraints. Similarly, the values used in this example are for illustrative purposes only.

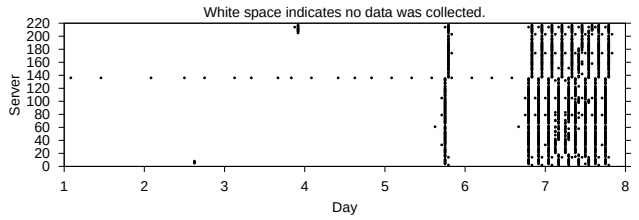

Figure 11: Data coverage in a consolidation data set

these 10 servers, and provide a recommendation on how to consolidate the remaining 90 servers using two weeks of data. Alternatively, the consultant could use only the first week of data (which still satisfies all three policies), in which case only servers 1-7 would be excluded.

Figure 11 shows a data set that clearly should not be used for providing a consolidation recommendation. Although the data set "passes" the duration requirement (loosely speaking), it fails both the frequency and the coverage requirements. In fact, all 220 servers would be excluded by the firm's quality requirements.

Our data assurance solution would benefit a consulting firm in several ways. First, the modules could be written (or customized) once by a single expert user, then used by all of the consultants. This would ensure that the firm's data quality policies are consistently applied in all server consolidation exercises. Second, the firm could establish policies for recording information about the types of imperfections encountered in the data collected in each customer engagement. This would enable a single expert within the firm to revise the modules as new imperfections are encountered, in order for all recommendations to continue to meet the data quality requirements. Since the data is typically stored in a database, our solution could easily be integrated into the firm's existing process (just as we described in Section 6.1 for SASU).

A third benefit is that selected modules could be invoked periodically during data collection at each customer site. For example, a consultant could set up a daily cron job to automatically verify that the data collection is proceeding in a manner that will yield data that meets the firm's quality requirements. If we consider the second data set, the consultant could have been informed via email after the first day that there were data quality problems, as data from only a single server was being collected, and not at the expected frequency. The consultant could take immediate action to keep the project timeline from slipping. The firm's data quality expert could also develop modules to provide more specific feedback on potential causes for data quality issues so to further reduce the time needed to address them. For example, on day 5 in Figure 4, the consultant could have been informed that collection had not been reestablished for servers 1-7, and thus the consultant could focus on them.

7. RELATED WORK

While many disciplines are aware of imperfections in data and have developed techniques for addressing the problem, in the IT management domain, we are aware of only ad-hoc solutions. Although many of these techniques are relevant to the IT management domain, challenges exist in applying them, such as identifying those techniques that are suitable yet do not add unnecessary complexity. Our work is focused on these challenges. As data assurance requires a cross-disciplinary approach, it is infeasible to describe all relevant related work, or even list all domains that have attempted to address data quality issues. We mention, however, existing work that can be applied to the IT domain.

Applied mathematics and statistics offer many techniques that may be leveraged but, as noted in Section 5.1.2, challenges remain in applying these techniques. For identifying possible imperfections, we could, for example, utilize techniques for identifying outliers or influential observations [15].

When data is missing, there are numerous filling techniques, several of which we mentioned in Section 4. In order to apply the appropriate filling techniques, we may need to use additional techniques to determine if the data are missing at random, missing completely at random, or not missing at random [14, 13, 10]. Any or all of these techniques could be implemented as modules in our solution. Additional work is needed to determine which techniques are most useful (and when), in dealing with imperfections in IT data. Understanding the impact of imperfections on the robustness of statistical techniques is an important aspect of this process [18].

In data mining, poor data quality can hinder the mining process [2]. Anand *et al.* developed a data cleansing kernel for data mining [2]. While the functionality provided by their kernel could assist with cleansing IT data, we believe our solution is more flexible, as it does not require consumers to run on a specific platform. Our work also provides domain knowledge for data quality issues in enterprise IT environments, which is outside the scope of Anand *et al.*'s work. Furthermore, our solution can provide a centralized view of the imperfections in IT data, which facilitates greater sharing of domain knowledge.

Researchers in many domains, including social and health sciences [1, 8] are aware of imperfections in the data they use. Addressing data quality issues has been done in many domains (e.g., [19]). Finding and correcting imperfections in data with direct business uses has received significant attention. For example, Experian offers a service to perform tasks such as address validation and de-duplication of records.[8] We have not, however, seen much formalism applied to IT system utilization data, which is why we have been investigating this area.

Quantifying the quality of a data set is important, particularly as we move towards more automated uses of IT data. Pipino *et al.* describe a number of types of quality metrics that could be used [16]. Our approach uses one of these, namely, a ratio, which we evaluate along multiple dimensions (e.g., completeness of the data). Since quality is an important aspect in many other domains, we expect there are many other quality assessment techniques that we could leverage. We plan to investigate these in future work.

Finally, data assurance requires knowledge about imperfections. We've adopted a rule-based approach as the rules are simple, intuitive, and yield repeatable results, characteristics we wanted in our solution. Their main disadvantages include that they must be maintained, and they are inadequate for representing some types of knowledge and they require knowing ahead of time this knowledge. Research on capturing experience-based "deep-smarts" [12] is useful

[8] `http://www.experianintact.com/`

for addressing the former, while statistical methods could address the latter.

8. SUMMARY

Automated management tools are playing an increasingly important role in the management of enterprise IT computing environments. Many of these tools and systems are data driven, yet, as we have shown, IT data contains many imperfections. These imperfections can lead to inappropriate automated decisions that have significant technical and business consequences. Despite these effects, the poor quality of IT data has received only limited attention, likely because IT automation is placing new demands on IT data and few automated solutions exist today. Where data imperfections has been encountered, ad-hoc quick-fix solutions have been employed. However these limit the sharing of domain knowledge, impact the robustness of automation, and impose unnecessary development/maintenance costs.

To address these problems, we examined the types of imperfections that exist in IT data, and their causes. In this paper, we reported on this examination and discussed the resulting implications. We developed a data assurance software layer that provides mechanisms to systematically detect and correct the imperfections. These mechanisms can be programmatically invoked and customized to the specific needs of data consumers (e.g., people and automated management tools). Further, our solution can be easily extended to provide additional functionality. We have described our solution and two case studies that demonstrate the benefits of our approach. In one case, using our approach, a capacity planner of Shared Application Server utility now spends $\approx$ 3x less time on data quality issues while managing $\approx$ 5x more applications and searching for 9 rather than 1 imperfection type. In the other, a hypothetical company specializing in server consolidation who improve the accuracy of their recommendations and the likelihood of winning contracts by using our approach.

Our work makes three significant contributions. We raise awareness of this emerging and significant problem in enterprise computing that will become more obvious as automation is increasingly adopted. Second, we provide domain knowledge about the nature of some common imperfections in IT data, and thus provide a foundation for establishing generally acceptable practices for using imperfect IT data. Finally, we present our initial implementation of a data assurance layer. These contributions benefit data consumers, data providers, and developers of IT monitoring tools and automation tools.

We believe data assurance for IT data represents a new and important area of work, and presents many open issues and research opportunities. Some of the unanswered questions include: what mechanisms can be used to quantify uncertainty? what techniques are required to automatically cleanse other types of data? what meta-data is required to enable this cleansing? how can we build systems and management applications differently so as to lessen the occurence of imperfections? We invite other researchers to help us address these many open issues.

9. REFERENCES

[1] P. Allison. *Missing Data*. Sage Publications, Thousand Oaks, CA, 2001.

[2] S. Anand, B. Scotney, M. Tan, S. McClean, D. Bell, J. Hughes, and I. Magill. Designing a kernel for data mining. *IEEE Expert*, 12(2):65–74, March-April 1997.

[3] T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31:307 – 327, 1986.

[4] G. Box and G. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, CA, 1970.

[5] H. Brown. Protecting against nonrandomly missing data in longitudinal studies. *Biometrics*, 46(1):143–155, March 1990.

[6] C. Chatfield. *The Analysis of Time Series: An Introduction*. Chapman and Hall, sixth edition, 2004.

[7] T. Chou. *The End of Software: Transforming Your Business for the On Demand Future*. Sam's Publishing, Indianapolis, IN, 2004.

[8] P. Diggle, P. Heagerty, K.-Y. Liang, and S. Zeger. *Analysis of Longitudinal Data*. Oxford University Press, Oxford, UK, 2002.

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1, 1999.

[10] D. Heitjan and S. Basu. Distinguishing missing at random and missing completely at random. *The American Statistician*, 50(3):207–213, August 1996.

[11] G. Latini and G. Passerini. *Handling Missing Data*. WIT Press, 2004.

[12] D. Leonard and W. Swap. *Deep Smarts: How to Cultivate and Transfer Enduring Business Wisdom*. Harvard Business School Publishing, Boston, MA, 2005.

[13] R. Little. A test of missing completely at random for multivariate data with missing values. *Journal of the American Statistical Association*, 83(404):1198–1202, December 1988.

[14] R. Little and D. Rubin. *Statistical Analysis with Missing Data*. John Wiley and Sons, Ltd, New York, NY, 1987.

[15] D. Pena. Influential observations in time series. *Journal of Business and Economic Statistics*, 8(2):235–241, April 1990.

[16] L. Pipino, Y. Lee, and R. Wang. Data quality assessment. *Communications of the ACM*, 45(4):211–218, April 2002.

[17] R. Sauers, C. Ruemmler, and P. Weygant. *hp-ux 11i tuning and performance*. Prentice Hall, Upper Saddle River, NJ, 2004.

[18] R. Tweedie, K. Mengersen, and J. Eccleston. Garbage in, garbage out: Can statisticians quantify the effects of poor data? *Chance*, 7(2):20–27, Spring 1994.

[19] P. Westerman. *Data Warehousing: Using the Wal-Mart Model*. Morgan Kaufmann, San Francisco, CA, 2001.