Obfuscatory obscanturism: making workload traces of commercially-sensitive systems safe to release

Charles Reiss
University of California, Berkeley
Berkeley, California, USA
charles@eecs.berkeley.edu

John Wilkes, Joseph L. Hellerstein
Google
Mountain View, California, USA
{johnwilkes,jlh}@google.com

***Abstract*—Cloud providers such as Google are interested in fostering research on the daunting technical challenges they face in supporting planetary-scale distributed systems, but no academic organizations have similar scale systems on which to experiment. Fortunately, good research can still be done using traces of real-life production workloads, but there are risks in releasing such data, including inadvertently disclosing confidential or proprietary information, as happened with the Netflix Prize data. This paper discusses these risks, and our approach to them, which we call *systematic obfuscation*. It protects proprietary and personal data while leaving it possible to answer interesting research questions. We explain and motivate some of the risks and concerns and propose how they can best be mitigated, using as an example our recent publication of a month-long trace of a production system workload on a 11k-machine cluster.**

I. Introduction

Many in industry want to guide academic work to increase its relevance to their company's technical challenges, and believe that giving academic researchers access to relevant workloads would help both groups. Many of the largest clusters today are used for commercial workloads that support large-scale Internet services, which have different challenges than scientific and academic environments. Without guidance from data such as traces, researchers must make assumptions, but these may be at odds with the actual workloads. For example, traces from Hadoop MapReduce analytics clusters at Facebook and Yahoo showed that small jobs dominated the workload [1]. Reducing the response time of these small jobs was much more important than making small improvements in the throughput of the large ones, an insight that was contrary to a widely-shared intuition and previously available benchmarks. This would not have been obvious without access to traces of the real systems.

Traces of production systems are widely used within industry. Unfortunately, release of these datasets is rare, because they may contain sensitive information, or provide indirect evidence from which sensitive information can be deduced. Most prior work with sanitizing production traces deals with privacy concerns. Although privacy is a serious problem, it is not the only issue. Indeed, some traces have no privacy component, but are still sensitive because of business secrets that may be leaked, directly or indirectly.

This paper reports on techniques we developed to remove commercially sensitive information from production traces in a way that has little or no impact on the value of the traces to researchers. We call this process *obfuscating* the trace; it is a generalization of *anonymization*, which is frequently used to describe the removal of personal information and replacing the names of various components with hard-to-guess stand-ins.

TABLE I
Key obfuscation techniques for our trace.

Data type	*Examples*	*Transformation*
open	time, CPI	unchanged
strings	job name, user name	hashed
enumerated constants	priority, scheduling class	ordered (mapped to unique integers)
resource sizes	cores, RAM	rescaled (divide by max, limit precision)
other	constraints	(special)

The methodology described in the paper is quite general. However, we use the following scenario to ground our discussion. We recently prepared a cluster scheduler workload trace of a current commercial system for release to the public. The trace contains information about *jobs* submitted by *users*; each *job* is made up of one or more *tasks*, each of which has associated *requirements* and *resource usage* data. The client names stored in the original traces are of internal services or employees, and the tasks correspond to batch workloads or services that handle many external users. As a result, we are not really concerned about privacy: although the trace includes data about services that deal with data about external customers, the system being traced touches relatively little personal information. The bigger issue is that the company would prefer that we not accidentally reveal certain details about the hardware and software used in the cluster.

Obfuscation is surprisingly hard to do well. One factor that makes it difficult is that outsiders can use information from outside a trace to discover its secrets. This information can be simple facts about the world—like that "people often search for their own name" or "many programs request CPUs in units of a complete core". It is not possible to enumerate all such discoveries, so the problem is one of managing risk.

A. Contributions

In this paper, we present concerns that companies have about the risks of releasing traces from production systems. To address these concerns, we introduces mechanisms for obfuscating traces in a way that has little impact on the value of the traces to researchers. We applied these mechanisms to traces from Google production clusters, and have published 30 days of sanitized production data from a Google cluster.

The remainder of the paper is structured as follows. Section II surveys prior work. Section III summarizes why companies suppress certain information in production traces, and section IV discusses best practices for trace obfuscation. Section V describes difficulties inherent in trace obfuscation and some of the perils of hiding too much information. We illustrate the choices we made for our trace in section VI and our conclusions are contained in section VII.

II. Prior work

Several traces of real systems have been released to the public. Almost any trace has some issues that prevent its release in its raw form; this section lists a few examples, and what was done about these issues.

Traces captured from academic contexts may not be subject to commercial concerns, but careful consideration to the amount of personal identifying information (PII) is still required. Depending on the type of trace, this may be a major issue or simply not a problem – e.g., resource usage data from a scientific batch processing facility is largely unrelated to personal activities.

Well-studied classes of traces include filesystem accesses and network traffic. Several network traces from academic and research environments are distributed to the public [2]–[4]. These provide the public with a collection of traces, typically gathered from academic and research sources. The obfuscation in these traces ranges from removing HTTP request and response bodies (in an HTTP trace) to replacing all web page URLs and usernames with their opaque identifiers (in an home web access trace). Regardless of the degree of obfuscation, the LBNL-hosted Internet Trace Archive [2] requires that "archive users agree to not perform traffic analysis aimed at circumventing the degree of privacy present in the trace".[1]

Recent network traces have enjoyed more systematic approach to anonymization (such as described by [5]). Since these traces are used to evaluate systems that work on raw network traffic, these techniques focus on transforming network traces containing at least packet headers to "equivalent" network traces that omit private information. Most notably, this requires omitting raw IP addresses, but more subtle channels also need to be removed, such as TCP timestamps (which can identifying machines [6]) or HTTP response size patterns (which can identifying web sites [7]).

Several traces of network filesystem activity have been created by researchers (such as [4], [8]–[10]), usually from their own installations. Some of these traces are even from commercial installations [11]. In obfuscating these traces, it has usually been considered sufficient to replace each component of filenames and each username with a distinct identifier, but this may still leak some information about directory structures, file types, and so on.

Research grid computing installations [12] and traditional supercomputing installations [13] have made batch job-scheduler traces available. These traces typically include only relatively coarse-grained information about resource demands and usage. Typical obfuscation—if any—consists of replacing user, group and (possibly) job names with unique identifiers. Users of these systems do not generally feel the need to hide the fact that they are using them, or the types of programs they are running. Many of the datasets that have been released to the general public are limited purpose: for example, a collection of disk failure information [14] was released to allow others to reproduce the results of a paper on the topic.

Although research based on commercial workloads is not infrequent, there are few publicly-available commercial datasets and so analyses of commercial datasets are usually performed within the providing company, or by private arrangement with a research group. For example, the first author's (academic) research group has obtained batch workload traces from some large Internet companies, but these were available only under an agreement to not share the trace widely.

Research labs of companies will sometimes release data about their own systems, which are less commercially sensitive than systems running end-user workloads. For example, storage-system traces were made available for some research systems at HP Labs [15], but almost no examples of traces from commercial systems, whether running production code or benchmarks. One of the reasons offered was a fear of "giving away a commercial advantage"; another was "getting the system managers to agree – and they saw no point in releasing such traces" [16].

III. Reasons to obfuscate

There are several threats that obfuscation is meant to prevent. The most publicized are privacy risks: that the trace might reveal some embarrassing information about an individual, which may have legal consequences and cause public-relations difficulties. Another threat is the accidental release of proprietary information to competitors or customers. Another is leaking information that a malicious (or aggressive) competitor might use to damage the trace provider business through financial, political, or technical means – for example, a trace might accidentally reveal how to "game" one of the provider's services. This list is by no means exhaustive; we expand a little on a few of them below.

A. Competitive concerns

There are several types of trade secrets that a company may want to avoid revealing through traces. Exact policies differ between companies; there is no single formula. Our examples are intended to illustrate common concerns that make releasing commercial traces challenging.

[1]One of the larger issues for the community is the eagerness with which some people pursue reverse engineering trace data to see what can be found, and then publish the result, without necessarily considering the broader consequences. Non-disclosure agreements are sometimes used in an attempt to reduce the risk of this happening. They have other drawbacks, but these are outside the scope of this paper.

One class of secrets is information about unreleased products. When leaked, this can create substantial media attention, bypass carefully-planned marketing campaigns, force premature release (or delays), and allow competitors to undermine a new product, resulting in lost revenue. The resulting damages can be millions of dollars. Less frequently publicized, but still potentially bad, is information that is only likely to be useful to competitors or malicious third-parties. For example, these groups may be interested in inferring the company's supply chains, algorithms used for handling abuse, or the exact versions of hardware and software used internally.

Protecting information about a new product may be non trivial if prototypes of it are using shared infrastructure services that are being traced. This is not easy to bypass by releasing only traces of systems that serve no unannounced products – beside the difficulty of determining whether this is happening (it may not be known), or manually selecting parts of larger workloads (it may be too hard at scale), some products may not be acknowledged publicly for many years (if ever), so any traces old enough to meet the requirement will likely be long out of date.

Companies often want to avoid giving their competitors performance targets to aim for. Knowing that certain performance goals are practical would allow competitors to better allocate their engineering resources. For example, the existence of Google's MapReduce system is public, but releasing precise performance information would likely create an instant benchmark for other MapReduce implementations. This effect can be seen with the TeraSort benchmark: one year after Google released their time (68 seconds), Yahoo's Hadoop engineers had approximately equalled that time [17]. While there are benefits to releasing explicit performance targets, companies want to make such decisions explicitly.

Precise information about hardware and software could permit disruption of a company's business. For example, companies running large scale services may have chosen certain hardware components several years in advance of their production deployments, and may be committed to buying these particular components to gain price discounts. Given that they acquire or replace (at least) tens of thousands of servers and pieces of related equipment per year, they may account for a substantial portion of the entire market for some hardware components, and thus a potential target. A malicious competitor could increase costs by bidding up the price or causing an artificial shortage of selected, critical components, such as FLASH chips, or a particular kind of processor. Supply-chain management is hard enough without complicating it by introducing knowledgeable antagonistic agents.

Another concern is vulnerabilities to external attack in hardware or software. Since large-scale services tend to have relatively homogeneous hardware, firmware, and software platforms, each of these components is a high-value target. To make attacks more expensive to perform, service operators want attackers to have to guess the configurations and waste resources attacking systems the operator does not have.

Since it could allow the inference of performance targets, supply chains, and exact hardware configurations, information about the actual resources of each physical machine is often considered sensitive (e.g., the amount of RAM, number of cores, the network bandwidth). Even worse is information that might reveal the total size of a company's computing resources, as that may implicitly provide financial and capability data that a company would prefer to keep secret from its competitors, customers, and even suppliers.

B. *Privacy obligations*

The most obvious reason to obfuscate a trace of an online service is because end users may have provided (or generated) data with the expectation it would remain private. For example, these services may manage e-mail addresses, word processing documents, and web viewing and searching history. To alleviate regulatory concerns and preserve user confidence, service providers typically promise not to release any personally identifiable information (PII). Such information is varyingly defined to include information that "can be used to contact or identify [users]" [18] or that "personally identifies [users] ... or can be reasonably linked to such information" [19]. Frequently, laws and regulations attempt to enforce similar requirements.

Identification cannot be prevented by simply omitting names, social security numbers, postal addresses, e-mail addresses, etc. Violations in privacy from anonymized' database releases can arise from surprising correlations between the released data and publicly available data about people [20], [21]. For example, gender, birth date and 5-digit zip code uniquely identify around 63% of Americans [22].

However, many interesting traces, such as ours, are intended to characterize hardware and software and their behaviors, not end users. Such traces do not need to include any information correlating user requests to each other or identifying the specific resource that an end user was served by. The primary personal privacy concern comes from the messiness of the data: logs intended to show technical information may include portions of requests, database keys, file names, etc. Unfortunately, the need to remove such information generally makes it impossible to release freeform logs without extensive (and time-consuming) normalization of their contents.

IV. Obfuscation strategies

In this section, we discuss a few common strategies used to protect or remove sensitive information from a trace. There are basically three techniques: *culling* out parts of the original information and excluding it from the obfuscated version, *subsetting* to only a portion of the total data available, and *transforming* the original data in some manner that is hard to reverse-engineer, such as providing only *aggregate* (summary) data.

- *Culling*: only certain parts of the original trace data (e.g., fields in a trace record), need be included - the remaining data is omitted, or culled. This is often used on traces of end user actions, to eliminate PII. Because raw trace data

may contain labels that are user or product names or even excerpts of processed data, obfuscation almost always requires limiting the types of data in the released trace and culling (or aggressively transforming) any free-form text fields. Culling also helps make traces more compact.
- *Subsetting*: traces can also be obfuscated by selecting only a part of the available data (e.g., only certain clusters from the ones a company runs). This can be a productive strategy since most researchers are more interested in rich and representative data than complete data.
- *Transformation*: instead of deleting or ignoring information, it can be included in a different form (e.g., names could be sorted and then replaced by their position in the list; a text string can be replaced by its hash). The data items that occur in trace records can be thought of as defining columns. For each column in the raw data, there are numerous possible derived columns which preserve some information in the source column (such as equality between entries) while discarding at least some sensitive information.
- *Aggregation*: a special kind of transformation is to replace the actual data with a summary. For example, a distribution of job inter-arrival times, rather than the actual starting times.

A practitioner should select the operations trace consumers should be able to do (such as 'comparing resource measurements') and use these techniques to provide some minimal information sufficient to do so.

A. Comparison-preserving obfuscations

Probably the most common obfuscation transformation is to hide the actual values of names or labels, such as user names. The trick is to preserve the relationships between the values, such that a set of records describe work submitted by the same user. There are several common transformations for this purpose, including:

1) Put all visible labels for a column in some information-free order (such as random order or the order they first appear in the trace); assign them unique numbers in this order; include only the numbers in the output trace.
2) Take a keyed cryptographic hash (such as HMAC-SHA256) of the original label using a secret key that is unique over some space (e.g., the datatype).
3) Encrypt the values using non-randomized encryption (such as AES-CBC [23] with an initialization vector derived from keyed cryptographic hash of the encrypted value) using a key that is unique for the data type (an approach like this is implemented by HP Labs' DataSeries [24] library).

Approach 1 produces compact labels and is ideal when the space of all possible labels is easily determined in advance. Approach 3 has the advantage of being reversible by the trace providers, but it produces much larger identifiers (especially if everything is padded to hide length information). Approaches 2 and 3 have the advantage of not requiring an explicit label-generating phase and so are better suited to labels with many possible values.

In any case, trace producers should make an explicit decision about what columns of labels should be matched to, or correlatable with, other columns. Those where the trace consumer should be able to identify equality relationships should use the same transformation, and those which the trace consumer should not be able to correlate should use an independent transformation—different cryptographic keys or list of labels.

When the trace consumer should be able to identify more than equality relationships within a column, a different type of transformation is required. For example, to allow inequality and ordering comparisons across discrete values, while hiding the actual values, one can gather all the values for the column, sort them, and replace each value by its index into this list. This is likely to be a useful for parameters like software and hardware versions. This technique may also be applicable to parameters whose absolute numerical value is meaningful to performance—such as the number of disks on a machine—but restricts the kind of analyses that can be done (e.g., this cannot readily support determining the mean load on a disk drive).

B. Continuous measurements

Continuous-valued measurements are much more problematic to obfuscate. There are several properties of such information that a trace producer may want to preserve, such as:

1) whether two values differ by a large amount and in what direction;
2) the approximate ratio between any two values;
3) the approximate magnitude of each value; or
4) the presence of small differences between values;

Information of type (2) or (3) is likely to be most useful for researchers since they are straightforward to use for replay. To attempt to provide only ratios, one can linearly rescale the values in some uninformative way: for example such that the medium datum of that type in the trace is 1. To avoid providing information of type (4), the precision can be reduced, e.g., by rounding values to a small number of digits.

A common proposal is to rescale continuous measurements in order to hide sensitive hardware configuration information. Unfortunately, it is often difficult to reveal ratios without also providing approximate magnitudes, which may be sensitive information (e.g., the distribution of the number of cores per CPU in a cluster of machines). If someone examining the trace can estimate the true value of any one sample, then they can use that information to reveal the approximate magnitude of all samples in the trace.

For resource consumption in particular, a task might be using a certain number of some discrete resources like disk drives or CPU cores. A natural measure of the usage of each of these discrete resources is continuous: e.g., CPU-seconds used per second or disk read bandwidth achieved, so it might be thought that simply rescaling the consumption would be good enough. Not so. It is likely that a disproportionate share of programs will fully utilize all of a discrete resource such as

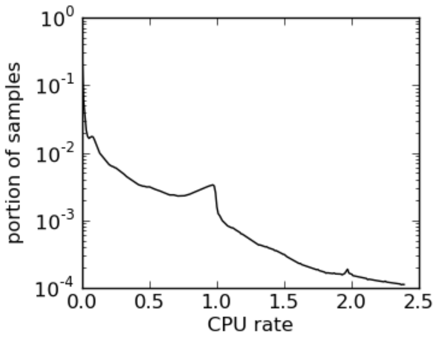

Fig. 1. Histogram of task CPU utilization measurements from a cluster at Google. The x-axis represents the number of CPU-seconds per second a task used on a machine during a 5-minute measurement period. The data includes all available measurements of all tasks on the cluster over many days, except that measurements greater than 2.4 CPU-seconds per second are omitted. These measurements were bucketed into 0.01 CPU-second per second bins; the y-axis represents the portion of measurements falling into each bin.

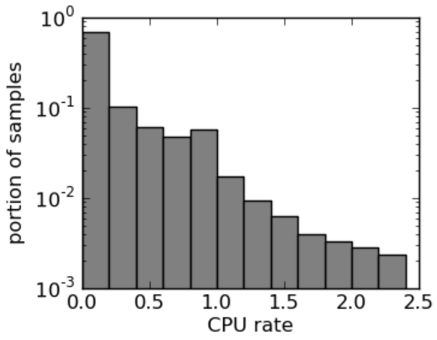

Fig. 2. The same data as in Figure 1, grouped into bins that are 0.2 CPU-seconds per second wide instead of 0.01. There is still a distinguishable event at 1.0 cores.

a CPU core, so this may be used to reveal the actual resources counts.

For example, figure 1 shows an excerpt from the histogram of task CPU utilization measurements on a cluster at a Google. Each measurement represents the number of CPU-seconds per second used by a task on a single machine over the measurement time period, which is 5 minutes. It is easy to see from this histogram where "1 core" is because a disproportionate number of task samples show one core being fully utilized.

This effect might be mitigated by providing much less precision in the provided ratios or averaging the utilization over longer periods of time—but a lot of precision may need to be culled. For example, the bump near 1.0 cores is still clear when CPU usages are only given to a precision of one-fifth of a core.

Averaging over long periods of time will diminish the effect because fewer samples will experience full utilization of a unit of the resource. Like removing precision in magnitude, such averaging would clearly diminish the utility of the trace, since it would require hiding the existence of tasks with sustained CPU bursts.

To prevent this type of attack, trace providers might try to distort the distribution. Unfortunately, many examples of combining two resource usages may reveal the shape of the distortion. Any useful distortion will preserve the property that values which are originally close will be close in the resulting distribution. Given this, one can bucket the distorted distribution into N fixed sized buckets and approximate the inverse of the distortion function in each bucket by a list of N numbers $D[i]$. Knowledge that some (distorted) measurement S is the sum of $a, b, \ldots$, reveals information about D: $D[S] \approx D[a] + D[b] + \ldots$. With sufficiently many such samples and a guess for any $D[x]$, this information produces an overdetermined system of approximate equations. This system can framed as an optimization problem to minimize some measure of the error and solved using standard techniques.

Thus, trace producers cannot rely on distorting measurements if they also provide aggregations of the measurements. This severely limits the information that can be provided since implicit aggregations—for example, the capacity of machines—are ubiquitous in any trace that measures both the incoming workload and its effects.

One could imagine that there are ways to add random noise to flatten the distinctive bumps in the distribution. Unfortunately, the noise might need to be quite large. A trace consumer can remove the noise by combining similar values together: for example, one may guess that tasks within a job have similar resource utilizations; making it possible to identify multiple (noisy) samples of a particular value. The samples can be averaged together to produce a low variance estimate of the mean.

C. Timeseries data

Traces usually contain data about the times of events. These data are particularly concerning for obfuscation because timing data from one source is easy to correlate with timing data from another source (finding external data sources to join against could be difficult for other data types). High-precision timing information is likely to allow correlating supposedly obfuscated requests to each other and perhaps obtaining precise performance benchmarks. To prevent easy correlation with external events, a trace provider can shift all the times to some new origin, and perhaps rescale the result, so that absolute timings are not available. Relative timings can be obfuscated by providing only low-precision timing information. To prevent anyone from obtaining aggregate relative timing information (as might indicate performance), the precision must be low, and/or noise should be added to the raw times first: the number of times a quantity "crosses over" a rounding boundary will provide an estimate of the mean if the shape of the distribution of the times can be guessed.

Even low-precision timing information risks correlating the trace to real-world events that may affect the workload. For externally visible services, outsiders might be able to correlate spikes or outages in a service to events in the trace. For example, the GMail service could be identified data based only on a correlated spike in abnormal task termination events and the date of a publicized outage. This effectively leaks information about which service this is, even though its name may have been obscured. Even without a event like this, it still may be possible to distinguish between different types of services based on diurnal or weekly patterns.

D. Hierarchical data

Many traces contain fields that are hierarchical. Common examples are filenames and IP addresses. Filenames and IP addresses sharing a prefix are likely to be related to each other and many analyses would benefit from this information. Releasing hierarchical information needs to be done with caution. For example, previous work on obfuscating packet traces [5], [25] has found that network traces contain scans across the IP space in sequential order. Combined with a few known IPs, this would easily permit discovery of a large number of IP mappings. Similar problems may exist for other hierarchical fields.

To preserve hierarchies, but suppress the names, one strategy is to break each of these hierarchical-name columns into multiple columns, where each synthetic column is a prefix of the original. For example, to obfuscate the filename "/usr/bin/scp", one could divide it into three ("/usr", "/usr/bin", and "/usr/bin/scp"), and transform each of these using an equality-preserving scheme. If necessary, the obfuscated trace might be culled to include only the prefixes. If the ordering of items is likely to reveal sensitive information (as with IP addresses), then the trace provider should explicitly check the trace for scans through the name hierarchy.

E. Sampling-based strategies

Traces can also be obfuscated by subsetting data sources (e.g., providing only a short-duration trace, or only a subset of the available systems or resources). This is common, especially in older traces, because of limitations on the quantity of information that could be gathered, stored, delivered, and processed.

Subsets can be picked to suppress sensitive information. For example, concerns about supply-chain disruptions can be mitigated by providing a trace from machines that are not representative of the company's newer acquisitions or future provisioning plans. Some services will span most of a company's infrastructure; for these services, selecting only a subset will yield an incomplete trace that cannot (for example) be replayed easily. Nevertheless, for many uses, providing fuller information for a subset of the actual machines is likely to be more useful than less precise information for the whole fleet of machines.

In addition to taking a subset of machines or jobs, one could also provide only aggregated data. That is, instead of providing records for each observed machine, job, task, request, etc., the trace provider might only provide descriptions of groups of machines, tasks, and jobs collectively. This mitigates concerns that are based on finding out information about particular jobs, users, or machines types. One form in which such aggregate data could be provided is as a synthetic trace or trace generator. This technique is proposed [26] as a way to produce exportable benchmarks from proprietary traces.

Aggregation limits the kinds of analyses that can be done on traces – in particular, on better ways to do the aggregation. The best that can be hoped for is that a trace synthesized from the aggregates will induce similar behaviors on the system as did the real one, but this is hard to test. For example, if the aggregation involves describing clustering of tasks with "similar" requirements, then a fair amount of information has already been suppressed; it is also likely that the clustering process is itself imperfect, and omits some important correlations. A synthetic trace produced from these cluster descriptions would have the same problems.

F. Differential privacy

Differential privacy [27] is a measure of the privacy provided by a computation based on how much the addition or deletion of a single data record can affect its output. A computation is *differentially private* if the possibility of getting a particular result changes less than a given threshold when a record is added or removed from the dataset. This threshold can be thought of as a privacy budget and can be divided among many computations to get more results and still satisfy a higher-level privacy goal. Because of this composability, differential privacy can be provided nearly generically by building on top of fundamental operations that add carefully chosen noise. To hide the presence of any individual record, these operations must be aggregates, but can be as general as "count the number of records such that a user-supplied function is true".

Probably the most attractive feature of differential privacy is that a trace producer does not need to guess which questions the trace should answer. Systems (such as [28]) have been proposed where within a privacy budget, trace producers could allow researchers to write arbitrary queries against the raw data and get automatically obfuscated results. The privacy budget makes this less attractive because it requires that only to a limited set of trace consumers be granted access, and all their queries must contain aggregates. Also, many competitive concerns are not addressed by the differentially privacy guarantee which, by design, preserves aggregate values. Even for personal privacy concerns, there is a practical problem of how to set the privacy budget since it may not be clear how many records need to be masked to prevent the identification of one person's activity.

V. Some perils of excessive obfuscation

Obfuscation can be taken too far. The most common problem is missing information. For example, [29] uses traces to evaluate the effectiveness of caching in a batch system; it could not have been done on a trace summary like that proposed by [30] which only reflects data sizes and compute times.

This section discusses a few of the difficulties that may arise when using an obfuscated trace.

A. Unavailable information

All traces are incomplete in some manner, possibly just because some desirable information was not readily available from the system being traced. When researchers produce their own traces, they may be able to do follow-up work to collect or approximate this extra information, but for traces released by third parties, applying such remedies is likely to be impossible.

Many real workloads are "half-open" loops: part of the workload is generated because prior tasks finished; other parts appear asynchronously. HTTP requests are an example [31]: some resources (e.g., images) are requested only because they are referenced by others (web pages).

Job- and task-level workloads are likely to have this flavor, too: users may use higher-level tools (such as [32]–[35]) to write workflows on top of a lower-level scheduler from which the trace is extracted. Unless the obfuscation is done with care, the workflow's dependency information is easy to cull by mistake.

Enough information may be present in the original raw trace to make reasonable estimates of dependencies. One could look for similar service names, user names, filenames, job names, binary names, etc. High-quality replay might involve processing those "hints" to find likely dependency chains. But, much of this information is likely to be suppressed or provided in a form that makes such analysis difficult to verify.

B. Missing semantics

Researchers analyzing traces frequently use knowledge of the purpose of the programs that appear in a trace. For example, this knowledge is important to argue that researcher's proposal would actually affect metrics that its users care about. Similarly, researchers modelling traces want to verify that their models are capturing attributes that are universal and not simply artifacts of one company's implementation.

Understanding the purpose of jobs within traces of shared infrastructure services is difficult. Standard labels, like job names, are not likely to survive obfuscation. The trace may provide many exportable hints of job purpose, like priorities, resource requests, and filenames, but these are imprecise and subject to company-specific quirks (such as defaults or approval requirements).

A researcher might use these hints to infer new scheduler constraints (such as deadlines) or to predict the workload. For example, proposals to automatically scale up and down allocations to a service (for power [36] or cost [37]) depend on short-term workload predictions. These systems could train a workload model using the trace, but the researcher would not know whether the model learned properties that are universal (such as end-user request patterns) or quirks of the company (such as periodic automated jobs being released at a particular time).

In short, it is helpful to consider the purposes to which traces will be put when making them available. If the goal is to foster work on improved scheduling algorithms (say), then the traces should include information to aid both replay and understanding of the workload.

VI. Obfuscating our trace

In November 2011, we released a trace of 29 days of job requests and usage data for a medium-sized production compute cluster at Google that contained about 11 000 computers, using the techniques described in this paper. Our most difficult competitive concerns were addressed by subsetting: to avoid revealing the size of the company's computing "fleet", we choosing a single cluster; to avoid releasing sensitive machine configuration information, we chose an older cluster whose machines did not have unusual hardware; to avoid most concerns about revealing application performance or structure we chose a cluster with a varied workload – one with many important services but not dominated by an especially sensitive service such as ad serving.

We took time to explain and verify the trace. Since we do not use a publicly available scheduler, we provided explicit documentation about the meaning and origin of each of the trace fields [38]. Before releasing the trace data, we checked it for internal consistency and against an alternate internal source of the trace information. These checks identified some bugs in our transformation code and our monitoring infrastructure that would have made the trace difficult to interpret, and might have been impossible for outside users to discover. One goal was to support experiments on job-scheduling, so we made sure to provide information about jobs that started and finished outside the time window of the trace, but excluded computers that were statically assigned to particular uses or users. (This caused a small infidelity for jobs that spanned two classes of machine, but the effect was around 0.003% of the load in the cluster.)

We used a combination of the techniques discussed above to transform the trace fields. For scheduling constraints on quantities like version numbers and spindle counts, we sorted the used values for each machine attribute and assigned an integer to each value. We then normalized all the constraints (equality or inequality comparisons against attributes) to use these integers. For example, if machines in the trace had version 2.4, 2.6, and 3.0 of some software, we represented these versions as 1, 2, and 3, respectively. A constraint like '`version >= 2.6`' would be obfuscated to '`opaque > 1`'; and '`version != 2.4 && version != 2.5`' to '`opaque != 1 && opaque != 0`' (using 0 since 2.5 never appears on a machine).

Text fields were encrypted using a keyed cryptographic hash (HMAC-SHA256) with a different key for each field type, derived from a keyed cryptographic hash of the field name and data type using a single master key for the trace. We hid the identity of particular services by hashing job and user names in this fashion, but to allow correlation analyses we provided *normalized job names* that stay the same across job re-executions.

For resource-size data such as capacity, usage and scheduler request sizes, we rescaled the resource units so the maximum observed machine capacity in the trace was 1.0, and deliberately limited the amount of precision we provided so that the intervals between possible values would not trivially identify a "unit" quantity. Despite this, we recognized that our rescaling of usage information was likely to be ineffective for at least some types of resources (such as CPU usage) – but we did so consciously.

We also provided some information in unobfuscated form, such as cycles per instruction and memory accesses per instruction measurements, since we did not believe it would otherwise be useful.

We believe that the balance we struck was able to meet the needs of our research colleagues and our own requirements for restricting some commercially-sensitive information.

VII. Conclusion

Removing private and competitive information from traces without damaging the utility of the trace is challenging. Ultimately, all obfuscation techniques must strike a balance between supporting interesting analyses and the risk of revealing confidential information. A significant problem is the lack of certainty: one cannot prove that everything has really been sanitized.

Privacy concerns make it difficult to release precise information about user activities because the user information that makes those traces unique also poses a privacy risk. Competitive concerns create different challenges (e.g., releasing aggregate data may be undesirable), as well as some similar ones (e.g., job names may be as sensitive as customer names).

Applying the most effective obfuscation schemes will severely limit the analyses that can be performed. Nevertheless, this may be appropriate when fairly complete data is important (e.g., for evaluating schemes for balancing work between datacenters). Otherwise, it may be more useful to provide a more complete set of data from a smaller source than a small amount of data from a broad range of systems.

As always, there's no free lunch. Academics should recognise that the reasons for industry's hesitations are real, and important. If industry would like to foster higher-quality, more relevant research into their problems, they will need to find ways to accept some of the inherent risks, and make more data accessible. We hope that this paper will serve to provide guidance on the likely consequences, and best practices for doing so.

References

[1] M. Zaharia *et al.*, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proc. 5th European Conf. on Computer Systems*, 2010, pp. 265–278.

[2] "Internet Traffic Archive." [Online]. Available: http://ita.ee.lbl.gov

[3] "LBNL/ICSI Enterprise Tracing Project." [Online]. Available: http://www.icir.org/enterprise-tracing/Overview.html

[4] Storage Networking Industry Association, "SNIA IOTTA repository." [Online]. Available: http://iotta.snia.org/traces/list

[5] R. Pang *et al.*, "The devil and packet trace anonymization," *SIGCOMM Computer Communication Review*, vol. 36, pp. 29–38, January 2006.

[6] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Trans. Dependable Secur. Comput.*, vol. 2, pp. 93–108, April 2005.

[7] Q. Sun *et al.*, "Statistical identification of encrypted web browsing traffic," in *Proc. IEEE Symp. on Security and Privacy*, 2002, pp. 19–30.

[8] D. Ellard and M. Seltzer, "New NFS tracing tools and techniques for system analysis," in *Proc. USENIX Conf. on Large Installation Systems Administration*, 2003.

[9] M. G. Baker *et al.*, "Measurements of a distributed file system," in *Proc. 13th ACM Symp. on Operating Systems Principles*, 1991, pp. 198–212.

[10] J. Hartman, "Sprite traces." [Online]. Available: http://now.cs.berkeley.edu/Xfs/SpriteTraces/sprite_traces.html

[11] E. Anderson, "Capture, conversion, and analysis of an intense NFS workload," in *Proc. 7th Conf. on File and Storage Technologies*, 2009, pp. 139–152.

[12] "Grid Workloads Archive." [Online]. Available: http://gwa.ewi.tudelft.nl/

[13] "Parallel Workloads Archive." [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/

[14] "The computer failure data repository (CFDR)." [Online]. Available: http://cdfr.usenix.org

[15] "HP Labs Storage Systems Program tools and traces." [Online]. Available: http://www.hpl.hp.com/research/ssp/software/

[16] J. Wilkes, Private communication.

[17] O. O'Malley and A. C. Murthy, "Winning a 60 second dash with a yellow elephant." [Online]. Available: http://sortbenchmark.org/Yahoo2009.pdf

[18] "Quantcast privacy policy." [Online]. Available: http://www.quantcast.com/how-we-do-it/consumer-choice/privacy-policy/

[19] "Google privacy policy." [Online]. Available: http://www.google.com/privacy/privacy-policy.html

[20] A. Narayanan and V. Shmatikov, "Myths and fallacies of 'personally identifiable information'," *Communications of the ACM*, vol. 53, pp. 24–26, June 2010.

[21] ——, "Robust de-anonymization of large sparse datasets," in *Proc. IEEE Symp. on Security and Privacy*, 2008, pp. 111–125.

[22] P. Golle, "Revisiting the uniqueness of simple demographics in the us population," in *Proc. 5th ACM Workshop on Privacy in Electronic Society*, 2006, pp. 77–80.

[23] M. Dworkin, "NIST SP 800-38A: recommendation for block cipher modes of operation—methods and techniques," *National Institute of Standards and Technology, US Department of Commerce*, 2001.

[24] E. Anderson *et al.*, "DataSeries: an efficient, flexible data format for structured serial data," *SIGOPS Operating System Review*, vol. 43, pp. 70–75, January 2009.

[25] J. Xu *et al.*, "Prefix-preserving IP address anonymization: Measurement-based security evaluation and a new cryptography-based scheme," in *Proc. 10th IEEE Int. Conf. on Network Protocols*, 2002, pp. 280–289.

[26] A. Ganapathi *et al.*, "Statistics-driven workload modeling for the cloud," in *Proc. 5th IEEE Int. Workshop on Self-Managing Database Systems*, 2010, pp. 87–92.

[27] C. Dwork, "A firm foundation for private data analysis," *Communications of the ACM*, vol. 54, pp. 86–95, January 2011.

[28] F. McSherry, "Privacy integrated queries: an extensible platform for privacy-preserving data analysis," *Communications of the ACM*, vol. 53, pp. 89–97, September 2010.

[29] G. Ananthanarayanan *et al.*, "Disk-locality in datacenter computing considered irrelevant," in *Proc. 13th Workshop on Hot Topics in Operating Systems*, 2011.

[30] Y. Chen *et al.*, "The case for evaluating MapReduce performance using workload suites," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2011-21, Mar 2011.

[31] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open versus closed: A cautionary tale," in *Proc. 3rd Symp. on Networked Systems Design and Implementation (NSDI)*, 2006.

[32] C. Chambers *et al.*, "FlumeJava: easy, efficient data-parallel pipelines," in *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation*, 2010, pp. 363–375.

[33] A. Thusoo *et al.*, "Hive: a warehousing solution over a map-reduce framework," *Proc. VLDB Endowment*, vol. 2, pp. 1626–1629, August 2009.

[34] C. Olston *et al.*, "Pig latin: a not-so-foreign language for data processing," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2008, pp. 1099–1110.

[35] P. Couvares *et al.*, "Workflow management in Condor," in *Workflows for e-Science*, I. J. Taylor *et al.*, Eds., 2007, pp. 357–375.

[36] A. Krioukov *et al.*, "NapSAC: design and implementation of a power-proportional web cluster," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 102–108, 2011.

[37] B. Trushkowsky *et al.*, "The SCADS director: scaling a distributed storage system under stringent performance requirements," in *Proc. 9th USENIX Conf. on File and Storage Technologies*, 2011, pp. 12–12.

[38] "Google cluster-usage traces: format + schema (2011.11.08 external)." [Online]. Available: http://goo.gl/5uJri