Workload-Aware Database Monitoring and Consolidation

Carlo Curino
curino@mit.edu

Evan P. C. Jones
evanj@mit.edu

Samuel Madden
madden@csail.mit.edu

Hari Balakrishnan
hari@csail.mit.edu

ABSTRACT

In most enterprises, databases are deployed on dedicated database servers. Often, these servers are underutilized much of the time. For example, in traces from almost 200 production servers from different organizations, we see an average CPU utilization of less than 4%. This unused capacity can be potentially harnessed to consolidate multiple databases on fewer machines, reducing hardware and operational costs. Virtual machine (VM) technology is one popular way to approach this problem. However, as we demonstrate in this paper, VMs fail to adequately support database consolidation, because databases place a unique and challenging set of demands on hardware resources, which are not well-suited to the assumptions made by VM-based consolidation.

Instead, our system for database consolidation, named Kairos, uses novel techniques to measure the hardware requirements of database workloads, as well as models to predict the combined resource utilization of those workloads. We formalize the consolidation problem as a non-linear optimization program, aiming to minimize the number of servers and balance load, while achieving near-zero performance degradation. We compare Kairos against virtual machines, showing up to a factor of 12× higher throughput on a TPC-C-like benchmark. We also tested the effectiveness of our approach on real-world data collected from production servers at Wikia.com, Wikipedia, Second Life, and MIT CSAIL, showing absolute consolidation ratios ranging between 5.5:1 and 17:1.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Relational databases; Distributed databases; H.2.7 [**Database Administration**]: Metrics

General Terms

Experimentation, Measurement, Performance

Keywords

consolidation, multi-tenant databases

1. INTRODUCTION

With the advent of outsourced computing and storage in the form of public clouds, as well as the rapid rise in the number of web, mobile, and enterprise database applications, it is now common for a single data center within an organization to deploy hundreds or thousands of individual relational database management systems (DBMSs). For example, one large telecommunications company with which we are familiar has more than 20,000 DBMS instances deployed in its internal infrastructure. Oftentimes, each database is deployed on a dedicated server, with the machine provisioned for the peak load that is expected to be placed on the database. However, it is unlikely that all the DBMS instances in a data center will be hit by peak loads at the same time. In practice, most databases have natural ebbs and flows, occasional unexpected events, and a certain degree of statistical correlation (or lack of correlation) with other databases in the same data center. Over-provisioning and uncorrelated loads provide the opportunity to consolidate servers onto fewer physical machines. For example, by analyzing data we gathered from production database systems run by Wikia.com, Wikipedia, Second Life, and MIT CSAIL we found that a 5.5× to 17× reduction in the number of database servers is possible. This kind of savings can reduce hardware expenses, lower the administrative burden, and consume less energy [10].

The process of consolidation involves analyzing the load characteristics of multiple dedicated database servers and packing their workloads into fewer physical machines, reducing the resources consumed without changing the application performance. Of course, consolidating servers is not new a new idea, and has been the driver for widespread deployment of virtual machines (VMs) in data centers. VMs have been particularly successful at consolidating services that are not data-intensive, as well as testing/development environments. However, as we will show in this paper, consolidating databases is harder because DBMSs make strong assumptions regarding the characteristics and performance of the underlying physical system (e.g., that the buffer pool will always be located in RAM) and are designed to use all the resources they are given. This makes it hard for existing VMs or administrators to estimate the true resource requirements of complex DBMS software when trying to identify opportunities for consolidation. Furthermore, the use of a VM per database leads to significant redundancy (e.g., multiple independent disk operations for log and data writes, multiple copies of the OS and DBMS, etc.). For these reasons, we believe that VM-based consolidation is appropriate only when hard isolation between databases is more important than cost or performance.

We have identified two key challenges that need to be tackled when building a database consolidation system. First, tools are needed to accurately monitor the resource utilization of each database and to estimate the utilization of a combined set of databases. Second, algorithms are needed to choose which databases should be combined and placed on which hardware, given hundreds of databases and physical resources to choose from. This second challenge is difficult because, unlike many other software systems, data-intensive computations are not easy to migrate

SIGMOD'11, June 12–16, 2011, Athens, Greece.

between physical machines[1], thus, common load balancing strategies that perform migration in reaction to change in load cannot be applied. As a result, stable assignments of databases to physical machines that can run unchanged for days are more suitable.

This paper presents the design, implementation, and experimental evaluation of *Kairos*,[2] a system that consolidates multiple OLTP and Web database workloads on to a shared hardware infrastructure. These workloads are amongst the most common and profligate in enterprises, and are amenable to consolidation; handling large, scan-intensive OLAP workloads requires different modeling techniques, which are not discussed here.

Kairos begins with a set of independent database workloads running on dedicated servers. The output is a consolidation strategy mapping workloads to physical nodes. After consolidation, each physical node runs a *single* DBMS instance that processes transactions on behalf of multiple databases, while meeting the application's performance requirements. In particular, in this paper, we use the throughput of a database before consolidation as an implicit service level agreement (SLA) on the performance the system should provide after consolidation. Extending the system to support latency-based SLAs would make an interesting future extension of our work.[3]

In this paper, we concentrate on scenarios where each logical database places a moderate but non-trivial load on the underlying system, which is what enables consolidation. We envision tens to hundreds of databases being consolidated onto a single server—for this reason we ignore the schema scalability problems that arise in extreme multi-tenancy scenarios where thousands of almost inactive database workloads with very similar schemas are consolidated onto a single physical server. Such problems have been been investigated in depth by others [15, 3, 14].

This paper makes three principal contributions:

1. Techniques to estimate resource requirements, particularly CPU, RAM and disk I/O, and models to estimate resource needs for combined workloads.

2. A method to analyze the resource consumption for each database over time in order to produce an assignment of databases to physical machines. Using the resource models mentioned above, we apply mixed-integer non-linear optimization to minimize the number of required machines and balance their load, without changing the perceived performance of the databases. The technique can be used to find the best assignment of databases to machines at time-scales ranging from hours to months.

3. Experiments on real-world load statistics from almost 200 production servers provided by Wikia.com, Wikipedia, Second Life and our lab. We find consolidation ratios (i.e., the ratio of machines used before and after consolidation) ranging from 5.5:1 to 17:1. We compare our approach of one database instance per machine with one virtual machine per workload, as well as one database process per workload. We show up to 12× greater throughput and consolidation ratios up to 3.3× higher when running TPC-C.

2. SYSTEM OVERVIEW

[1]This is particularly true for direct-attached storage, but remains valid for storage-area networks (SANs). In fact, even with a SAN, there are limits to the available I/O and to the number of physical machines that can be connected.

[2]The Greek god of opportunity, which seems an apt name for a system that relies on opportunism.

[3]One reason explicit SLAs were not our initial focus was that we gathered from interviews with a few consultants and administrators that database-level SLAs were rarely used, compared to application-level SLAs.

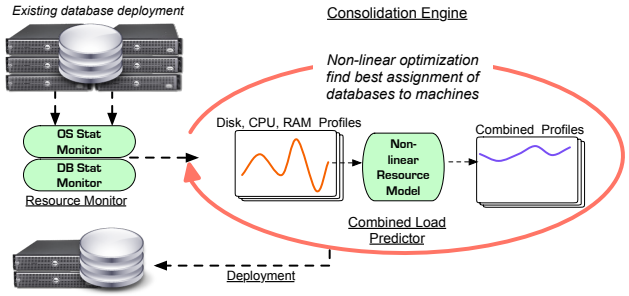

Figure 1: The architecture of the Kairos system.

Kairos takes as input an existing (non-consolidated) collection of database workloads, and a set of target physical machines on which to consolidate those workloads. The source and target hardware resources are allowed to overlap.

As shown in Figure 1 the key components are:

1. *Resource Monitor*: Kairos includes an automated statistics collection tool that captures data from the DBMS and OS to estimate the resource consumption of individual databases while running. This monitoring must be done without introducing any overhead. Estimating the RAM required by a database is particularly challenging and important; Kairos uses a simple, novel technique to gauge the working set size of a database—see Section 3.

2. *Combined Load Estimator:* Given the hardware resource consumption of individual databases running on dedicated hardware is a starting point, the next step is to predict how they interact when consolidated onto a single database server. Modeling the interaction is especially challenging for disk I/O because disk throughput is a complicated nonlinear function of the load, unlike with CPU or RAM which basically combine linearly. To solve this problem for the disk, we built a tool that creates a hardware-specific model of a given DBMS configuration, allowing us to predict how arbitrary workload mixes will perform on that configuration—see Section 4.

3. *Consolidation Engine:* Kairos uses nonlinear optimization techniques to find assignments of databases onto physical resources that: (i) minimize the number of machines required to support a user's workloads (ii) maximize load balance across the consolidated servers, and (iii) guarantee that every workload is assigned to a server without over-committing any server. Replication and other requirements on workload placement are naturally handled as additional constraints. The consolidation engine is described in Section 5, with implementation details in Section 6.

One way to think of Kairos is as a "consolidation advisor," which produces a static placement that database administrators (DBAs) can use to manually optimize their infrastructure. This procedure can be repeated periodically. Ideally, this process should be automated. We are in the process of working on a system to seamlessly migrate databases to handle this, as part of our Relational Cloud project [7]. For experiments in this paper, we manually implemented the consolidation strategies recommended by Kairos.

We note two important caveats about Kairos:

1. Our implementation is primarily for MySQL, though we believe the ideas we present generalize to other DBMS servers. The consolidation algorithm itself is more widely applicable, but the resource models used are based principally on observations and workload profiles from MySQL. In some cases, we have run additional experiments with PostgreSQL, demonstrating a slightly wider range of applicability of the ideas. To the extent possible, we point out where other database systems might differ from what we observed or where amendments to our techniques are required.

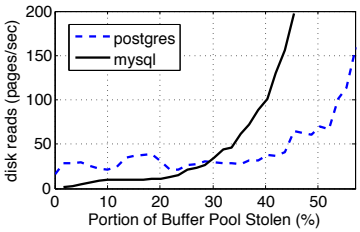

Figure 2: Buffer Pool Gauging

2. Consolidation is likely to increase latency and recovery time, because both are proportional to the load on the database system. Database consolidation, regardless of the specific consolidation approach, may not be appropriate for applications that are very sensitive to these effects.

3. RESOURCE MONITOR

The resource monitor queries the OS and DBMS running on each machine for statistics about CPU, RAM, and disk I/O, buffer pool utilization, and log flushes. This data is then used to predict the resource consumption of workloads if they are combined. For CPU, this task is straightforward because the CPU utilization of a consolidated workload is the sum of the CPU loads of its each of its constituent workloads, minus some small savings from reducing the number of OS and DBMS instances.

For RAM, the statistics provided by the OS tend to overestimate the actual resources required, as they can only report the total memory allocated to the database process, rather than the memory actually in use. This gap between the apparent and actual resource utilization is one of the primary reasons why VM-based consolidation does not work well with databases. This overestimate occurs because DBMSs have traditionally been designed to be the primary service on a machine, and thus attempt to use all available resources, even if they do not *need* those resources to sustain the same transactional throughput. For example, most databases will fill the buffer pool with pages from disk, even if the working set for the application is smaller than the buffer pool.

Thus, unless the application is actively accessing the entire buffer pool at maximum throughput, most DBMSs can deliver identical performance using less RAM. The challenge is accurately estimating the required RAM.

For disk, OS tools like `iostat` provide an easy way to estimate the disk I/Os of a single DBMS. The challenge is that databases often use unused disk bandwidth to proactively complete tasks—for example, when idle, MySQL/InnoDB aggressively flushes dirty pages from the buffer pool to reduce recovery time. Under most conditions, this flushing has no effect on transaction throughput, and the degree of flushing or I/O will change when multiple databases are combined together. We address the problem of estimating the I/O requirements of combined database workloads in the next section.

3.1 Buffer Pool Gauging for RAM Estimation

To operate efficiently, an OLTP DBMS needs to keep the *working set* of the application it is serving in main memory. Because it is difficult to estimate the working set size, it is common for administrators to devote nearly all the available RAM on the machine to the DBMS. Typical configurations include a very large buffer pool and no OS file cache (suggested configuration for most DBMSs including MySQL/InnoDB), or a combination of buffer pool and OS file cache (suggested configuration for PostgreSQL). In both configurations, after running for some time, all the memory accessible to the DBMS, i.e., the entire buffer pool and potentially the OS file cache, will be full of data pages. However, for many applications on modern servers with large amounts of RAM, the total amount of memory far exceeds the actual working set at any point in time—such over-provisioned applications are exactly those that are amenable to aggressive consolidation. Thus the challenge is to develop techniques that estimate the working set size of a database to determine if two or more databases can be consolidated together.

```
function bufferGauge(db, probeTable):
  tableLength = 0
  if db.tableExists(probeTable):
    tableLength = db.execute("SELECT COUNT(*) from ?", probeTable)
  else:
    db.execute("CREATE TABLE ? (id int, dummy char(?))",
      probeTable, PAGE_SIZE)

  scanLength = INITIAL_SCAN_ROWS
  while scanLength < MAX_SCAN_ROWS:
    if tableLength < scanLength:
      appendRows(db, probeTable, scanLength - tableLength)
      tableLength = scanLength

    for i in range(0, SCANS_PER_INSERT):
      db.execute("SELECT COUNT(*) FROM ? WHERE id < ?",
          probeTable, scanLength)
      sleep(READ_WAIT_SECONDS)
    scanLength += SCAN_INCREASE_COUNT
```

Figure 3: Probing Procedure

The first step in our gauging process is to determine if the system is over-provisioned. To do so we collect statistics about *OS-level disk reads*, and *DBMS-level buffer pool miss ratios*. This information tells us whether: (i) the working set fits in the buffer pool (the miss ratio of the buffer pool is close to zero), (ii) the working set does not fit in the buffer pool, but fits in the OS file cache (high buffer pool miss ratio is but very few physical disk reads), or (iii) the working set size exceeds the memory accessible to the DBMS, causing it to serve data from disk (high buffer pool miss ratio and many physical disk reads). Case (iii) indicates that memory is not over-provisioned, and that the total RAM accessible to the DBMS is actively used. n cases (i) and (ii), we need some facility to measure how much memory the database is actively using, either in the OS file cache, as in case (ii), or in the buffer pool, as in case (i).

We have discussed this subject with a number of database administrators who agree that it is common for OLTP/web databases to be over-provisioned in real-world installations. This motivated the design of a new technique, *buffer pool gauging*, to measure the working set size of an application.

Our technique issues SQL queries to an unmodified DBMS and observes internal statistics to estimate the database working set size. Pseudocode for our gauging procedure is shown in Figure 3. It works by creating an empty *probe table* in the database and growing it slowly, or reusing an existing probe table if we have performed gauging on this DBMS before. While adding to the probe table, we force the DBMS to keep its pages in the buffer pool by querying it several times for every round of inserts. The contents of the probe table and its query workload were chosen to force the data to be memory-resident without introducing additional logging or much CPU load: (i) the probe table has only a few large tuples that fit exactly into the page size, (ii) the queries are simple COUNT(*) aggregates over non-indexed fields, and (iii) there are no updates. While growing this table, we monitor disk reads that the DBMS performs using the statistics provided by the DBMS and the OS. By slowly "stealing" buffer pool/file cache space and monitoring the number of pages the DBMS reads back from disk, we are able to detect when we start to push *useful* pages out of the buffer pool/file cache. This is because queries from the legitimate user's workload cause evicted pages to be read back from disk.

Determining how quickly to grow the probe table (the `SCANS_PER_INSERT` parameter in Figure 3) is an interesting

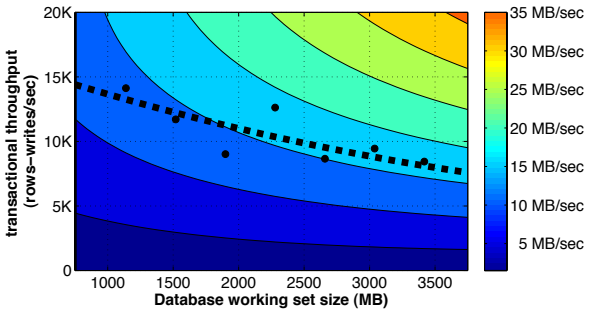

Figure 4: Disk Model for Our Experimental Configuration

question. On one hand, we want to to avoid affecting the overall query throughput of the system with our probing. On the other hand, if we grow the probe table very slowly it may take a long time to complete the gauging. In our current implementation, we used a simple adaptive strategy that accelerates the speed at which we inflate the probe table if there is no increase in the physical reads, while slowing down when we see even a small increase in the average number of physical reads per seconds over a short time window (the default in our tests is 10 seconds). In our tests the probe table growth speed can be as high as several MB/sec or as low as tens of KB/sec depending on the user workload. This allows us to avoid degrading the throughput of the database while maximizing the rate at which we can probe.

The frequency at which we query the probe table (`READ_WAIT_SECONDS` in Figure 3) also requires a careful balance: we need to guarantee that the probe table is queried frequently enough for the buffer manager to keep data in RAM, but we also want to avoid frequent queries that will only add to the CPU overhead. In our experiments, querying the probe table once every 1–10 seconds forces the DBMS to keep the probe data in RAM, while keeping the CPU overhead to under 5%.

We implemented this technique on MySQL and PostgreSQL and tested it using: (i) TPC-C, (ii) synthetic micro-benchmarks, and (iii) a benchmark we derived from Wikipedia. In the following discussion, we use TPC-C to discuss the effectiveness of our approach; in Section 7 we report a more thorough evaluation of the technique's effectiveness and impact on user performance.

Figure 2 shows the number of physical page reads/sec issued by the DBMS as we grow the probe table for TPC-C, scaled to 5 warehouses. The two lines present: (i) MySQL with 953 MB of buffer pool configured using O_DIRECT writes to bypass the OS file cache and (ii) PostgreSQL, configured with 953 MB of shared buffer, and using the OS file cache—filling the remaining 1GB of RAM. In both cases, the working set fits in the buffer pool, and we can "steal" up to 30%-40% of the buffer pool with a negligible increase in buffer pool misses (and thus of disk I/O). This matches our expected TPC-C working set size, which is around 120-150MB per warehouse. This allows us to detect that identical performance can be achieved by using a significantly smaller amount of RAM. To verify that this could not be done properly by the OS, we looked at Linux's reported "active" pages, which and found that all of the database buffer pool was shown to be active (1.7GB.) Thus, compared to OS metrics, we reduced the memory requirement estimate by almost a factor 2.8×. In Section 7, we report factors up to 7.2× in tests using the Wikipedia benchmark.

Our current prototype performed well in our experiments using MySQL and PostgreSQL on several workloads, and we thus believe this technique to be widely applicable. However, adapting it to a wide range of production environments requires significant engineering effort to deal with: (i) systems operating with multiple buffer pools (e.g., DB2), (ii) DBMSs which dynamically adapting buffer pool resources, (iii) cold databases with empty buffer pools, or (iv) quickly changing working set sizes.

4. COMBINED LOAD ESTIMATOR

In this section, we address the problem of estimating the resource requirements of a combination of several database workloads. For CPU and RAM, this problem is straightforward (once we have properly gauged the RAM requirements of each database): for each time instant we can simply *sum* the CPU and RAM of individual workloads being co-located. For disk, the problem is much more challenging.

4.1 Combined Disk I/O Performance

Predicting the disk performance of a set of database workloads is a hard problem, for a number of reasons:

1. DBMSs typically exploit unused disk bandwidth to flush dirty buffer pool pages back to disk whenever the disk is underutilized. Although this flushing is not required for correctness, it reduces recovery times, and may avoid mandatory writes due to future buffer pool misses or log reclamation. While using idle resources is in general a good strategy, it makes it very hard to estimate the minimum I/O resources required to achieve certain performance.

2. We have observed that disk I/O throughput grows sub-linearly with the workload size and the speed at which the DBMS is driven by user requests: more update operations will hit the same page at higher rates. Thus, each page that is written back includes writes from many transactions. Additionally, for larger data-set sizes, the ratio of log-writes to page flushes also changes.

3. Complex interactions between the DBMS, OS, and disk controller make it hard to predict how sequential or random the combination of a set of workloads will be.

4. There are a number of hardware and software configuration parameters that influence the I/O throughput a system can achieve, including group-commit timeouts, log-file sizes, workload speed and size, dirty page flushing policies, the number and types of disks, and caching in the OS and disk controller.

We can simplify the modeling problem because we are only concerned with modeling several consolidated databases inside a single DBMS instance, rather than the more complex problem of modeling several independent DBMSs running in one or more OSes.

A DBMS, regardless of the number of databases it hosts, coordinates I/O between those databases by: i) combining log writes from different workloads into a single sequential stream of log writes, which also leverages group-commit[4], and ii) delaying write-back operations of dirty pages for all databases and performing them in sorted order to reduce disk seeks. The result is a more efficient and predictable estimate of I/O throughput.

However, the complex behavior of disk subsystems still makes modeling the I/O requirements of a combined workload difficult. There are two ways one might attempt to address this problem: i) by building an analytical model of each component and algorithm affecting disk performance in the DBMS, or ii) by treating the system as a black box and experimentally deriving its transfer function.

In our experience the first strategy has two major drawbacks; first, it requires a detailed understanding of many DBMS internals, which are often not available for commercial products. Second,

[4]Some DBMSs support a separate log per database. This is useful when there is a separate disk per log, in which case our model can be applied to each disk independently. However, the general argument remains valid: the DBMS, even when writing multiple logs, controls the scheduling of log writes, and thus produces more sequential disk access patterns than a general purpose I/O scheduler.

any change in the algorithms or parameters, OS settings, or hardware requires modifying the model. We initially set out to build such an analytical model, but found that it only applied to very specific configurations. For these reasons, we adopted the second approach. We built a tool that automates the process of collecting experimental data from a live system and builds an *empirical model* that captures disk behavior.

Given a DBMS/OS/hardware configuration, our tool tests the disk subsystem with a controlled synthetic workload that sweeps through a range of database working set sizes and user request rates—this testing can be done as on offline process on a similar configuration, and does not need to interfere with the production database. The workload we use for this test is based on TPC-C, and represents a general OLTP workload. Our workload generator allows us to control both the working set size and rate at which rows are updated. At each step, the tool records the rows updated per second, the working set size in bytes, and the overall disk throughput in bytes per second. The result is a map of the system response to various degrees of load and working set sizes.

For the experiments in this section we collected over 7,000 data points on a test machines equipped with two quad-core Xeon 2.66 GHz CPUs, 32 GB of RAM and a single 7200 RPM SATA drive, running MySQL. The working set for the workload always fits in RAM, since this is the common configuration for the OLTP environments we are targeting.

Figure 4 shows a two-dimensional polynomial fit of this data—the actual data points are not shown for the sake of clarity. The X-axis shows the working-set size, the Y-axis shows the throughput in updated tuples per second, and the contours indicates the bytes written per second[5]. Increasing the rate at which rows are updated results in a non-linear increase in the aggregate I/O. Somewhat surprisingly, a larger working set also results in more I/O, again with a non-linear relationship. This is because when updates are spread throughout a larger working set, they are more likely to touch clean pages, resulting in more total dirty pages per unit of time and causing more pages to be written back. The thick dashed line shows a quadratic curve fit to the maximum disk throughput for each dataset size (the maximum throughput points are plotted as black circles). This is the point of disk-I/O saturation for our single disk configuration. Larger working set sizes yield lower throughput because they require in average more pages to be written back to achieve the same transaction throughput, and this experiment is disk bound.

This map is used to predict how multiple workloads will behave when combined. After a large number of exploratory experiments we observed (and later validated) that the combined throughput of several workloads respects the following property when the working set of the consolidated databases fits into memory:

Running multiple databases, with aggregate working set size X, at an aggregate row modification throughput (i.e., update, insert, or delete rate) Y produces the same disk I/O request rate as running a single workload with working set size X at update throughput Y.

This property holds for two reasons. First, in the steady state, no data needs to be read from disk since we assume that all working sets fit in memory. Second, the disk I/O is composed of: i) log writes, whose throughput depends only on the update rate Y and transaction log record size, which is roughly constant and small for typical OLTP workloads, and ii) dirty pages being written back to disk, which results in a similar mix of sorted random writes for either the single or multiple database case.

Therefore, to predict the amount of disk I/O that will be used when multiple workloads run inside a single DBMS, we monitor each workload for a period of time, collecting the working set size and the row update rate. By summing these parameters from all workloads, and using a profile like that shown in Figure 4 for the target host machine and database, we can predict how much disk I/O the combined workload will need. To the purpose of consolidation, it is only relevant to obtain a precise estimate for the high-load portion of the curve, as this is the part required to verify that the combined workload will not saturate the disk. Precision for very low disk load is not nearly as important, since it will not influence consolidation decisions anyway. As we show in Section 7, this model is significantly more accurate than the simple approach of summing the disk I/O of a set of consolidated workloads, especially for high-load situations where it reduces the estimation error by up to a factor of $32\times$.

In Section 7.5 we present experimental evidence that, for a broad range of workloads, the profile is independent of number of databases, database size and transaction types, and that it only depends on: (i) row update rates, (ii) the working set size, and (iii) the DBMS/OS/hardware configuration. This allows us to reuse profiles across a broad spectrum of workloads.

In our implementation we profile a machine by scanning the space of throughputs and working set sizes, testing each possible combination, this takes about two hours. We are investigating ways to speed up this process, including sampling fewer points, and building models incrementally. If a similar configuration has already been profiled, a small number of data points should be sufficient to adapt an existing profile.

5. CONSOLIDATION ENGINE

We now turn to the problem of determining which workloads should be combined together. The goal is to find an assignment that minimizes the number of machines, and that balances load across those machines as evenly as possible, while avoiding resource overcommitment. We model the problem of assigning workloads to servers as a mixed-integer non-linear optimization problem. The inputs to this problem are a list of machines with disk, memory, and CPU capacities, and a collection of workload profiles specifying the resource utilization of each resource as a time series sampled at regular intervals. We also allow the user to specify replication requirements for each workload, as well as pin individual workloads to specific machines. Machine capacities and resource utilizations are scaled appropriately to account for heterogeneous hardware.

Formally, we state the optimization problem as follows (these functions have been simplified for the sake of presentation; additional details are in Section 6):

$$\underset{x}{\text{minimize}} \quad \sum_{tj}(e^{(\sum_i C_{ti} * x_{ij})} * signum(\sum_i x_{ij}));$$

$$\text{subject to} \quad \forall i \sum_j x_{ij} = R_i;$$

$$\forall j \; max_t(\sum_i CPU_{ti} * x_{ij}) < MaxCPU_j;$$

$$\forall j \; max_t(\sum_i MEM_{ti} * x_{ij}) < MaxMEM_j;$$

$$\forall j \; diskModel(DISK_{ti}, x_{ij}) < MaxDISK_j;$$

$$\ldots$$

$$additional\ placement\ constraints$$

$$\forall i, j x_{ij} \in N; 0 \leq x_{ij} \leq 1$$

[5]We use a Least Absolute Residuals (LAR) second-order, polynomial fit of the disk I/O to build the disk model shown by the contour of Figure 4. We chose this model because it was accurate enough in our experiment to accurately predict disk I/O rates and in our consolidated workloads.

where $x_{ij} = 1$ if workload i is assigned to server j and 0 otherwise, and R_i is the number of replicas desired for workload i. $MaxCPU_j, MaxMEM_j, MaxDISK_j$, are the maximum amount of CPU, RAM and, disk I/O available at server j (these can be $< 100\%$ to allow for some headroom), and C_{ti} is the time series describing the utilization of resource C (CPU, RAM, disk) for workload i at time t.

Objective function: The primary goal of the objective function is to minimize the number of servers used for the consolidated workloads; our formulation achieves this by using the $signum$ function, which is equal to 1 when its input is > 0 and 0 when its input is 0 ($signum$ of a vector v is a vector v' where $v'_i = signum(v_i)$).

This model guarantees that any solution using $k-1$ servers will have a lower objective function value than any k server solution—if no workloads are assigned to a server the $signum$ function is zero, and the entire cost associated with that server is discarded. When at least one workload is assigned to a server, we evaluate the server's contribution by weighting its resource utilization over time C_{ti} using an exponential function[6]. This penalizes solutions that are unbalanced. In fact, for any given number of servers, our objective function has a minimum when the load is balanced. This corresponds to solutions for which the risk of saturation is minimized. The global minimum of our objective function is the most balanced solution among those using the smallest number of servers. Figure 5 shows a rendering of our objective function as a projection on a 2D plane, for a hypothetical scenario in which a 4-server solution is the optimum. The figure also shows that the solver we use increases the value of the objective function by a large factor when constraints are violated, as shown by the spike on the left side for less than 4 servers.

Constraints: The constraints guarantee the feasibility of the solution. Because each x_{ij} can only be 0 or 1, and because the sum of the x_{ij} values for a given i must equal R_i, we guarantee that each workload i has R_i replicas assigned to *different* servers. The CPU, RAM, and disk constraints guarantee that the combined load imposed on each server will not exceed the available resources at any moment in time—thus avoiding saturation and overcommitment of the servers. Notice that extending the set of constraints to include other resources (e.g., network, disk space) is straightforward. In this paper, we focus on CPU, RAM and disk I/O since these were the most constrained in the real-world datasets we obtained. In particular, for CPU and RAM the constraints are a simple maximum of the sum (matrix product) over time t, while for disk we use a non-linear function based on the disk model discussed in Section 4.

Since we compute the combination of resources over time, we introduce more non-linearities. However, this allows us to leverage time-dependent utilization patterns to achieve greater consolidation. To this end, we assume that the future resource utilization of a workload is strongly correlated with the past resource utilization. We validate this assumption using real-world data in Section 7.5.

Replication: Our model also allows us to handle replication and other workload placement issues by introducing additional constraints. The R_i parameter allows us to control many replicas of each workload need to be placed; we operate under the conservative assumption that the replica will consume the same amount of resources as the primary. However, if the input workloads are already replicated, we can use the actual load of the replicas.

In either case we need to guarantee that the consolidation engine will not place replicas on the same physical node. This requirement is achieved by adding constraints of the form: $\forall j \;\; x_{i'j} + x_{i''j} \leq 1$, which enforces that at most one of the two workloads i' or i'' are placed on a given target server j. Similarly we can support user requirements that pin a certain workload i' to a specific node j', by forcing $x_{i'j'} = 1$.

[6]We can use any linear combination of the resources, to favor balancing one resource over the other.

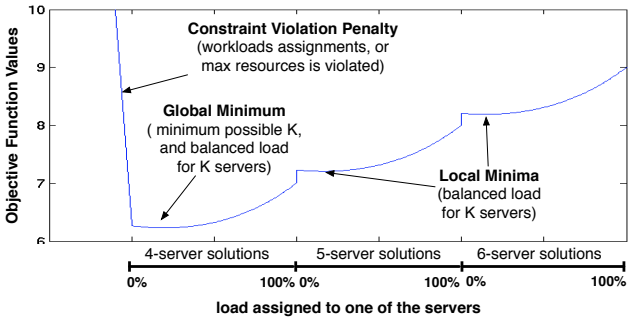

Figure 5: Simplified rendering of the objective function

The cost of optimization: Since both our objective function and some of the functions characterizing resource constraints are non-linear, and because the overall optimization problem has several local minima, we employ a general-purpose global optimization algorithm called DIRECT [16] to find a solution. This makes the search expensive, motivating us to find ways to improve the performance of the consolidation engine by exploiting the characteristics of our problem, as discussed in Section 6. We report the performance of our solver in Section 7.5.

6. IMPLEMENTATION & OPTIMIZATION

We implemented the monitoring tools discussed in the earlier sections in Java. They connect to the databases to be consolidated via JDBC, and use SSH to collect OS-level statistics. The consolidation engine is based on the implementation of the DIRECT algorithm provided by the Tomlab [12] solver library for Matlab. The data collection tools and output consolidation schemes have been tested against various combinations of MySQL and PostgreSQL, and various versions of Linux, running on x86 servers. We chose these configurations because they are popular in production systems; in particular, MySQL+Linux is used in all the real-world datasets we obtained.

Speeding up the consolidation engine: In the previous section, we described the non-linear programming approach we took to assign workloads to machines. The search for the global minimum is hard due to: i) the presence of local minima, ii) the non-linearity of the problem (forcing us to use a less efficient general-purpose global solver), and iii) the large number of variables and constraints. The key problem is the time spent by the solver considering solutions around local minima. This situation can be partially improved by controlling a parameter of DIRECT that determines the ratio of time spent in local versus global search. However, this risks compromising the quality of the load assignment solution, since not enough time is spent "polishing" the final solution around the global minimum. To address this problem, we developed an optimization that leverages our domain-specific knowledge.

The optimization attempts to reduce the number of variables by bounding the number of servers K that the optimizer has to consider. Suppose we can compute the minimum number of servers K' for which a valid solution exists. Then the solver can simply discard any solution using more than K' servers. While the value of K' is unknown, we can estimate reasonable lower and upper bounds for it. The lower bound is provided by a single-resource fractional solution that optimistically assumes that the workloads can be assigned fractionally to machines, and that each resource can

be considered independently. With these assumptions, the lower bound $\underline{K}$ can be computed as the maximum of the sum of each resource over time divided by the total available resources. A loose upper-bound $\overline{K}$ is the number of machines currently in use; better upper-bounds can be found by running cheap, greedy workload allocation strategies.

Since upper and lower bounds are typically not too far apart, we can binary search to determine the lowest value K' of K that leads to a viable solution. We start with our upper and lower bounds, and run the optimizer with the midpoint number of servers. We either terminate it as soon as it finds a valid solution, or consider K to not be feasible if it runs longer than a configured time limit. We then repeat on the upper or lower range, as appropriate, until we determine K'. We then re-run the solver, giving it a maximum of K' servers to allocate to the workloads and we do not stop until the optimal solution is found or a maximum optimization time is reached. Limiting the number of possible servers reduces the number of variables, and thus explores a much smaller solution space where many local minima have been discarded. This allows us to parametrize the DIRECT algorithm in favor of local searches to increase the quality of the final solution. In all the cases we tested, this approach leads to significantly lower running times than a direct application of the solver to the entire solution space—up to a $45\times$ reduction in running time. Near-optimal solutions are found in less than 8 minutes for problems with up to 100 workloads and 20 output servers.

Normalization and preprocessing of data: The above description simplifies the objective and constraint functions used in our non-linear optimizer for the sake of presentation. There are several additional details that we describe here for completeness. The first additional complication is that we limit the argument of the exponential objective function shown in Figure 5 to the $[0, 1]$ interval through normalization. This provides a function with a concave shape, and with reasonable numeric limits for the optimizer. Also we introduce weighting constants on each term in the linear combination of resources inside the objective function. These weights allow us to indicate that certain resources are more important to be balanced, while still preferring solutions with more servers over those with less less servers.

The CPU utilization reported by the Linux kernel is expressed as a percentage of one CPU core. Thus a value of 250 means that the system is using 2.5 cores. We first convert the percentages from heterogeneous machines to a "standard" core by scaling based on clock speed (ideally we would also scale based on CPU architecture). Then we convert the utilization to a fraction of a "target" machine. In our case, we assume the target machine has 12 cores. Thus, a CPU utilization of 250% would become $\frac{2.5}{12} = 0.208$.

When aggregating CPU and RAM from multiple workloads, we applied several other techniques:

CPU: Operating systems and databases each introduce some CPU overhead. When consolidating multiple workloads, simply summing the CPU utilization will double-count this portion of the load, since each machine we measure is running a separate OS and DBMS, while our consolidated solution has only one OS and one DBMS. To address this, the function that sums CPU usage in our consolidation engine removes a small, experimentally determined, fraction of CPU-load for each consolidated workload.

RAM: Our RAM model allows users to input a user-defined "scaling" factor that linearly scales down the measured RAM values. This can be used when buffer pool gauging techniques are not applicable. This is the case for the historical statistics we obtained from some of the organizations for this paper (Wikipedia, Second Life). For these workloads, we estimated an approximate 30% savings would be possible, based on manually examining a few production databases. Precisely determining this parameter requires significant domain/application knowledge.

7. PERFORMANCE EVALUATION

In this section, we begin by demonstrating the accuracy of our combined resource consumption models for CPU, RAM and disk, and testing our consolidation engine on a small-scale deployment in our lab. We then study the effectiveness of our consolidation algorithm on resource utilization profiles from almost 200 production servers from four organizations. Finally, we compare our approach to VM-based consolidation and validate that past load is a good predictor for future load.

7.1 Datasets and Experimental Setup

We experimented with two kinds of datasets: (i) *Database Workloads*, a series of synthetic and real workloads used to run fully controlled experiments, and (ii) *Real-World Load Statistics*, a set of historical statistics from production databases, used to verify the applicability of our consolidation engine. We could not test our entire suite of load profiling tools for these production databases, because we were not given access to the machines, but only the statistics that these organizations already collect.

Database Workloads: We used three main database workloads: (i) a synthetic micro-benchmark, (ii) TPC-C, and (iii) a benchmark derived from Wikipedia .

The synthetic micro-benchmark is used to verify that our resource models and consolidation engine function as expected. This benchmark contains five independent workloads that each operate on a single table, issuing a mix of updates and CPU-intensive selects (using expensive cryptographic functions). These workloads are designed so we can precisely control the amount of RAM, CPU and disk I/O consumed. To achieve this, we vary the working set size, the rate of selects, and the rate of updates. By adjusting these parameters, we generated five different workloads. Each workload has different time-varying patterns (e.g., sinusoidal, sawtooth, flat with different amplitude and period). The goal was to validate that Kairos properly predicts resource utilization for a very different set of workloads very than the one used to devise the resource models (based on TPC-C). It also tests the ability of the consolidation engine to detect opportunity for consolidation when multiple resource constraints make it challenging.

The second workload is based on TPC-C. We control the request rate and the working set size (by changing the number of warehouses clients access). TPC-C represents a typical OLTP workload, and we use it to validate the impact of consolidation on throughput and latency.

The third workload is based on Wikipedia. This benchmark models the actual workload on Wikipedia's database servers. We used the publicly available source code and data, along with a trace containing 10% of 4 months of actual HTTP requests served by Wikipedia's data centers, obtained from the Wikimedia Foundation. By combining this information with published statistics about the effectiveness of the many caching layers in the application [23], as well as private communication with Wikipedia engineers, we built a simulator that generates a workload similar to that observed by the backend MySQL database servers. We can scale this benchmark from a few tens of megabytes to the full 4 TB of the Wikipedia database by sampling an appropriate subset. This workload is transactional with 4 types of transactions, and models reads and edits of articles, watch list management, user logins, IP blocks, and a number of other features. On average 92% of the queries are reads and

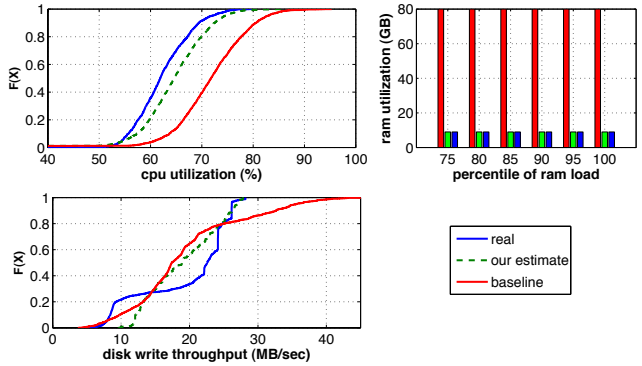

Figure 6: Validating resource models and consolidation engine

about 8% are writes. Tuple sizes range from 70 bytes to 3.6 MB (due to article text).

Real-World Load Statistics: We also validated our approach using real data obtained data from four organization:

- *Internal*: data from 25 servers that support general purpose computing from our lab's IT staff. This workload is a combination of production systems as well as test and development servers.
- *Wikia.com*: a collaborative publishing platform with over 34 database servers.
- *Wikipedia*: the popular free on-line encyclopedia. We obtained monitoring data from their Tampa, Florida database cluster, containing 40 database servers.
- *Second Life*: the data from 97 database servers powering the on-line virtual world.

The statistics were stored in the `rrdtool` format, used by open source monitoring tools such as Cacti, Ganglia, and Munin, and as recorded by the respective organizations, and thus they do not include our buffer pool gauging technique for accurately measuring RAM utilization. The data includes CPU, RAM, and (for some machines) disk I/O numbers as reported by Linux, averaged over different time intervals—ranging from every 15 seconds for the last hour to every 24 hours for the last year.

In our experiments, we use a 24 hour time window with samples every 5 minutes—this was the best compromise between length of observation and sampling rates (we obtained similar results on weekly and monthly data). We normalized the CPU data by considering the number of cores per machine and the CPU-speed where available. We have detailed CPU and RAM statistics for all machines, but we have disk I/O statistics for only a small subset. We used the available data to estimate the disk utilization on other nodes. All of the database servers in these datasets are running MySQL, with the exception of 1 machine in the Internal dataset running Postgres.

Hardware: The database servers used for our experiments have: *Server 1:* two quad-core Intel Xeon 2.66 GHz E5430 CPUs, 32 GB of RAM, and a single 7200 RPM SATA disk, and *Server 2:* two Intel Xeon 3.2 GHz, 2GB of RAM, and a single 7200 RPM SATA disk. Both machines run Linux 2.6.31 and MySQL 5.5.5-m3. Our target machine for consolidation of the real-world dataset is a server with 12 cores and 96 GB of RAM, which represents a higher-end class of machines used by two of our data providers. This type of server costs between USD $6,000–$10,000 at the time of writing, depending on exact the configuration.

7.2 Resource Models and Consolidation

To test the accuracy of our monitoring techniques (Section 3) and models for combined resource utilization (Section 4), as well as the ability of our consolidation engine (Section 5) to discover consolidation opportunities even in difficult scenarios, we set up a controlled experiment based on the five synthetic workloads described above. We varied several parameters to generate workloads with differing request rates, read/writes ratios, CPU/RAM/disk load, and request patterns.

The workloads used in this test were chosen to *barely fit* within a single physical machine, with individual resource utilization patterns that make it hard to detect the opportunity for consolidation due to the combination of constraints induced by multiple resources. We first use our monitoring tools to collect load statistics for individual workloads in isolation, then predict their combined load and compute a consolidation strategy with our engine.

Kairos successfully predicts that the workloads can be allocated together on a single server. We further validate the proposed consolidation strategy by physically co-locating the workloads and running them, while monitoring the consolidated system's latency and throughput. As expected, the workloads fit within a single physical server, yielding *identical* throughput and 95th percentile latency increases of less than 1 ms.

For each resource (CPU, RAM, and disk), we recorded the estimated and measured load over time. In Figure 6 we show the fraction of the total resource used by the combined workload in each time window estimated using our models ("our estimate"), measured on our server ("real"), and estimated using a straight sum of the operating system metrics ("baseline"). The baseline represents the simple approach to estimating the combined utilization. We plot the Cumulative Distribution Function (CDF) for CPU and disk and percentiles for RAM in Figure 6 (since the working set sizes are constant and the CDF would be a simple vertical line). The estimates for most heavily loaded windows are the most important because, as discussed in Section 3, we need accurate estimates when a resource is nearing 100% utilization.

This experiment shows the need for sophisticated resource models, and the accuracy of the models we proposed. In particular:

RAM: For these synthetic data sets, we control the actual working set sizes, which range between 512 MB and 2.5 GB, and are constant over time. The figure shows that buffer pool gauging (Section 3.1) accurately measures these working set sizes, while summing the OS statistics grossly overestimates the memory utilization as almost 9× the actual value of 9 GB.

Disk I/O: the CDF of disk I/O shown in Figure 6 shows that our disk model (Section 4.1) is very accurate at estimating the 75th-100th percentile of load, as in this range, the "estimate" CDF is very close to the "real" CDF. We have a maximum error of 800 KB/sec, while the baseline overestimates by 26MB/sec. However, our approach is only marginally better than the baseline for the central portion, and even worse than baseline in the bottom 30%. As noted above, we are primarily concerned with estimating the load on servers nearing 100% utilization—which is where the accuracy might affect consolidation decisions. Although it is not important to accurately estimate lower load percentiles in our application, if doing so was valuable, one could create a hybrid model that uses the baseline for percentiles below 30%.

CPU: our resource estimate factors out the fraction of CPU load introduced by additional copies of the OS and DBMS that are not needed when consolidated, and is thus able to reduce the estimation error to about 6%, compared to over 15% for the baseline.

Consolidation effectiveness: We now validate the claim that the consolidation strategies produced by our system are able to maintain the same performance as in the unconsolidated case. In Table 1 we show experiments in which we ran different workloads using TPC-C scaled to 2 and 10-warehouses and Wikipedia scaled to 100K pages. We vary the intensity of the workloads by changing

Table 1: Impact of consolidation on Performance

Test id (DBMS)	Dataset	Throughput w/o cons.	Throughput w/ cons.	AVG Latency w/o cons.	AVG Latency w/ cons.
Consolidation recommended					
1 (MySQL)	TPC-C (10w)	50 tps	50 tps	76 ms	98 ms
	Wikipedia (100K p)	100 tps	100 tps	12.7 ms	16 ms
2 (MySQL)	TPC-C (10w)	250 tps	250 tps	113 ms	180 ms
	Wikipedia (100K p)	500 tps	500 tps	43 ms	49 ms
3 (MySQL)	5× TPC-C (10w)	5×100 tps	5×100 tps	77 ms	110 ms
4 (MySQL)	8× TPC-C (10w)	8× 50 tps	8× 50 tps	76 ms	125.8 ms
	Wikipedia (100K p)	50 tps	50 tps	12.7 ms	19 ms
Consolidation not recommended					
5 (MySQL)	5× TPC-C (10w)	5×400 tps	5× 177 tps	77 ms	426 ms
6 (MySQL)	8× TPC-C (10w)	8× 100 tps	8×87 tps	77 ms	1180 ms
	Wikipedia (100K p)	100 tps	86.2 tps	12.7 ms	1047 ms

the rate at which user requests are generated. For experiments 1–4 our consolidation engine predicts that consolidation of the input workloads is possible since: (i) the combined size of the measured working sets fits in the buffer pool of our test machines, (ii) the estimated CPU and Disk I/O are well within the 90% max utilization. The results presented in the table show that the throughput is unaffected, and that while there is some increase in latency, it is only a few milliseconds. This is small compared to some normal fluctuations that occur. For example, when MySQL performs a checkpoint in order to garbage collect log files, the average latency can increase by as much as 150 ms. Experiments 5–6 show cases in which our consolidation engine suggested that the workloads can not be consolidated, due to memory or disk I/O limits. If we attempt to consolidate anyway the result is significantly decreased performance. Additionally, these experiments showed that the loss in latency and throughput is approximately uniform across the consolidated databases, suggesting that MySQL does a reasonable job of dividing resources amongst a number of databases.

7.3 Consolidation with Real-World Data

In this set of experiments, we measure the consolidation ratios and the load balance that our consolidation algorithm can achieve on the Internal, Wikia.com, Wikipedia, and Second Life datasets.

Figure 7 shows the consolidation ratios that our system finds for these four datasets, as well as for the combination of all workloads merged together, named ALL. We compare with three alternative strategies. The first is the current deployment without consolidation (*reference solution*).

The second is a single-resource greedy bin-packing approach (*greedy*). This algorithm considers only a single resource, and places each workload in the most loaded server where it will fit using a first-fit bin packer. We then discard final solutions that violate the constraints on the other resources. We repeat this packing once for each resource, then take the solution that requires the fewest servers.

The third is an idealized lower bound, in which we assume both that workloads can be assigned as fractions of their load, *and* that we can decouple usage of multiple resources (*frac./idealized*). As shown in Figure 7, our approach matches the idealized lower bound in almost every scenario. It consistently outperforms the greedy approach, achieving consolidation factors ranging from 5.5:1 to 17:1. The greedy algorithm cannot be applied in all scenarios, because it finds solutions that violate constraints on the other resources; in these cases, no result is shown. It also tends to find highly imbalanced configurations, while Kairos finds well balanced workloads.

Since the consolidation ratios depend on the configuration of machines being consolidated onto, we also confirmed that our engine substantially reduces the total resource requirements; for example,

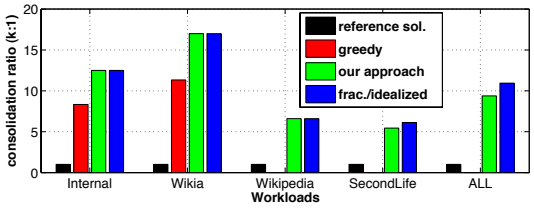

Figure 7: Consolidation Ratios for real-world datasets

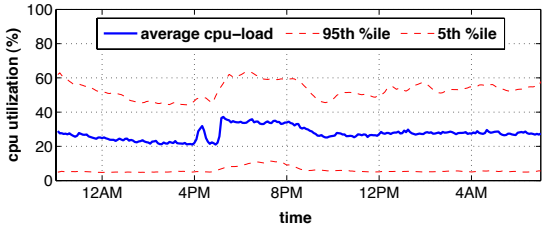

Figure 8: Aggregate CPU-load for 197 consolidated workloads.

in the ALL case, the original 197 servers has a total of 1419 cores versus 252 cores in the consolidated configuration.

We also measured the balance of resource utilization across servers. Figure 8 shows the average, 95th and 5th percentile of CPU-utilization for the 21 consolidated servers in the "ALL" experiments of Figure 7. The figure shows that the high and low utilizations are close, suggesting we have achieved good balance, and that the 95th percentile is far from the maximum utilization, suggesting low risk of saturation. Note that perfect balance, where all servers are close to the average load, is not achievable because: i) load cannot be assigned fractionally, and ii) there are multiple-resource constraints, thus, the most balanced solution for one resource might not be viable for other resources. Greedy solutions like the one we tested lead to much higher imbalance with 95th percentile load close to saturation on some machines.

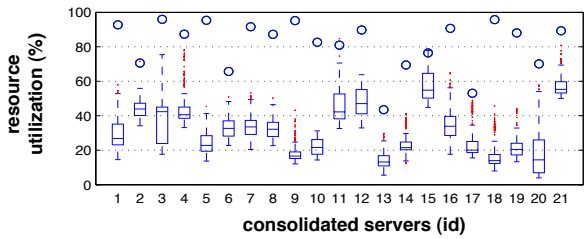

Figure 9: CPU load (error-bars) and RAM load (circles) for 197 real-world workloads consolidated on 21 servers.

Figure 9 shows the RAM and CPU utilization for these same 21 servers, showing the load across all time for each of the servers (rather than the aggregate load across all servers.) The values for each server are computed as the sum of the estimates of RAM and CPU of each workload consolidated on that server. The circles show the maximum RAM consumption on each server, while the CPU utilization is shown as a box plot. The top and bottom of the boxes represent the 25th and 75th percentile, while the line in the box indicates the median. The whiskers show the minimal and maximal values, and any outliers are plotted as points. Outliers are defined as points that are outside the interval $[q_1 - \frac{3}{2}(q_3 - q_1), q_3 + \frac{3}{2}(q_3 - q_1)]$, where q_1 and q_3 are the 25th and 75th percentiles, respectively. This figure shows that there is no easy way to further consolidate this load, since either RAM or CPU prevents combining any two servers. The figure also shows that the load is approximately balanced across the machines, and

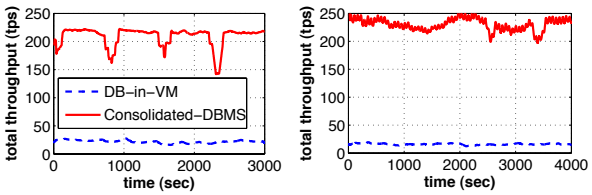

Figure 10: Hardware virtualization (fixed 20:1 consolidation)

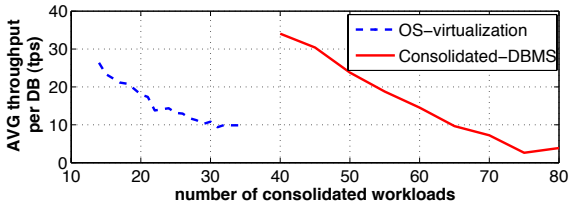

Figure 11: OS virtualization with varying consolidation levels

that we maintain a "margin of error" (set to 5% in this experiment) even for the most heavily loaded servers—this accounts for potential workload changes. Note that 5% of the machine's capacity represents close to 50% of the load of a single database, due to the nearly 10:1 consolidation ratio.

7.4 Comparing against DB-in-VM

In this section, we compare our database-aware consolidation to: i) hardware virtualization, where each database is running on top of its own operating system and virtual machine, using VMware ESXi, and ii) OS virtualization, where multiple MySQL instances run on a single Linux kernel, avoiding virtualization overhead (in this case, we do not use any additional virtualization software, and simply run each instance as a separate process.) We use multiple TPC-C instances to simulate users' workloads.

In order to investigate the performance of hardware virtualization using VMware ESXi, we show the throughput for a fixed consolidation level of 20 TPC-C instances in Figure 10. For this experiment, we found that VMware's RAM overcommitment feature made no difference, so we show results without overcommitment. The left-hand figure shows the throughput when all 20 workloads have the same request rate. The right-hand figure shows a skewed workload where 19 databases are throttled to one request per second, and 1 database runs at maximum speed. This shows that database consolidation can handle both uniform and skewed workloads, and yields better performance than hardware virtualization. Here, our approach provides between 6× and 12× higher throughput.

There are many reasons for this large performance difference:

- Running the workloads in separate database servers incurs a significant amount of redundant work for operations like log writes and log reclamations. Our approach more effectively exploits group-commit and coordinates disk requests across all databases, rather than each database making independent decisions. This allows our approach to achieve higher efficiency when using the disk.
- Our approach avoids allocating RAM for multiple copies of DBMS (MySQL ≈190 MB) and the OS (≈64 MB). VMware's "transparent page sharing" feature is intended to reduce this problem, however, it only reclaimed small amount of duplicated RAM.
- The virtual machine approach leads to more frequent and more expensive context switches, due to a higher number of processes.
- More effective utilization of CPU caches because more code and data is shared between workloads.

To remove virtualization overhead, we switched to OS virtualization, which is similar to container-based virtualization like Linux Containers or Solaris Zones. We then re-ran the experiment with a wide range of consolidation ratios. Figure 11 compares the throughput of OS virtualization and the Kairos consolidation system. In this experiment, we vary the number of consolidated workloads running on one machine, then measure the maximum average throughput achievable for each database, when the load is uniform across all databases. Given a specific target throughput, our approach yields consolidation levels 1.9× to 3.3× higher than OS virtualization. This translates to a significant reduction in hardware costs. The limitations of the VM approach remain when considering OS-level virtualization.

Finally we note that our approach and a VM- or OS-based approaches are not totally orthogonal. It is possible to combine the two approaches, running our consolidation inside a VM, to provide both benefits of virtualization and the performance of our approach.

Table 2: Impact of Probing on User's Perceived Performance

Target request rate	Throughput w/o gauging	Throughput w/ gauging	Latency w/o gauging	Latency w/ gauging
200 tps	200 tps	200 tps	8.57 ms	11.4 ms
600 tps	600 tps	600 tps	15.0 ms	19.1 ms
1000 tps	1000 tps	1000 tps	19.5 ms	23.9 ms
MAX	1923 tps	1689 tps	28.3 ms	32.1 ms

7.5 Additional Experiments

In this section, we show the results of additional experiments that measure the overhead of our techniques, the independence of our disk model from database size and workload type, and test our assumptions regarding workload predictability.

Impact of buffer pool gauging on performance: In Section 3.1 we showed that Kairos can accurately estimate the size of the working set of a system on both MySQL and PostgreSQL. The case we reported used a benchmark running at *maximum speed* on a database with a 953 MB buffer pool, growing our probe rate by 136KB/sec on average. This allowed us to limit the impact on performance to less than a 5% decrease in throughput and about 1ms additional latency, even though the machine was running at saturation. However, in typical scenarios we are going to face multi-gigabyte buffer pools and less busy machines.

To test that our approach can quickly measure large buffer pools without affecting performance, we tested a MySQL node with a 16 GB buffer pool. We ran the Wikipedia benchmark scaled to 100K pages (67 GB of data and a 2.2 GB working set). We forced the system to aggressively grow the probe table at an average rate of about 6.4 MB/sec. This allowed us to determine the working set size in about 37 minutes. Table 2 reports throughput and latency achieved by the system with and without the gauging for different user target request rates—we control user target requests rates by throttling the generation of requests on the machines simulating the Wikipedia clients.

The values reported correspond to the phase of gauging before we start stealing valuable user pages, at which point we quickly stop the probing and report the measured working set size. The MAX experiment is a control case, in which we test the impact of an aggressive gauging on a server that is already operating at saturation—and is thus not likely to be of any interest for consolidation.

Solver Performance: We also used the real world load statistics to test the efficiency of the optimization discussed in Section 6. The results vary with the dataset, but it consistently decreases the runtime of the solver, with the maximum reduction occurring with the Wikia.com dataset, where the runtime reduces by a factor of 45×—an equally consolidated and balanced solution is found in 44 sec instead of over 33 min in the unoptimized case. Running times for all individual consolidations were *below* 8 minutes, with

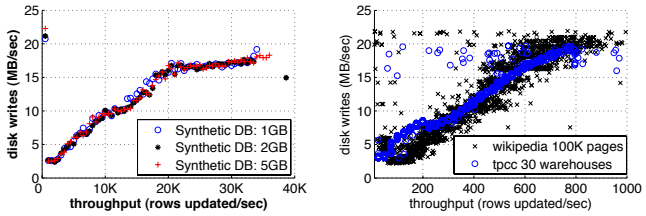

Figure 12: a) Database size does not matter, b) transaction type does not matter

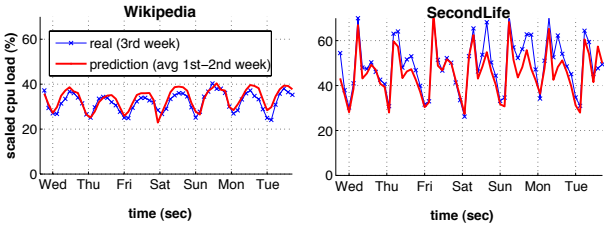

Figure 13: Predicting CPU utilization

the exception of the ALL workload, for which lack of RAM in our test machine prevented a proper measure (run-time heavily affected by swapping). A possible way to scale our solutions to handle tens of thousands of databases, consists in pre-grouping the input workloads, and solve the multiple consolidation problems independently. This might miss some small opportunity of consolidation (workloads across groups will not be considered for co-location), but it can be proven to scale indefinitely.

Disk Model Generality: In order to further validate our disk model, we need to show that: (i) the database size does not significantly affect disk I/O throughput (i.e., that a model learned on a dataset of size X still applies if that dataset grows to something much larger than X), and that (ii) the model is independent of the specific transaction being run, and thus that we can build a single disk model for a given DBMS/OS/hardware configuration without regard to the specific database workload.

In Figure 12a, we experimentally validate our hypothesis that the total database size does not influence the disk I/O throughput, but that only the working set size matters. For this experiment, we designed a synthetic test with a database of increasing size, and a workload that only accesses a random 512 MB portion of the database. The figure shows that the I/O throughput patterns for different database sizes are nearly identical, thus confirming the hypothesis that database size does not influence disk I/O.

In order to validated the hypothesis that the disk write throughput does not depend on the specific set of transactions being executed, but only on the overall number of rows updated per unit of time, and the overall working set size, we pick two very different workloads: TPC-C with 30 warehouses and Wikipedia with 100K pages; these databases have comparable working set sizes of about 2.2GB. Figure 12b shows the disk write throughput in MB/sec for a wide range of rows updated per second. It is clear that the two workloads, despite their major differences, impose almost identical pressure on the disk subsystem. Interestingly, while the average disk I/O rate for each rows-updated/throughput point is quite similar, Wikipedia has a significantly higher variance, due to the wider range of tuple sizes (from 70 bytes up to 3.68 MB). Note that the TPC-C database size is about 4.8GB while the Wikipedia one is over 67GB; this large difference further confirms the hypothesis tested above: database size has no noticeable impact disk I/O.

Past Performance as a Predictor of Future: Our final experiment tests whether past workload behavior is a good predictor of its future behavior. This is crucial to ensure that our computed strategy will satisfy the transaction throughput requirements over time.

We used our Wikipedia and Second Life largest real world load datasets to validate our hypothesis that workloads are predictable. We examine the total CPU utilization across all servers, as this is typically the most volatile measure. We divided the data into weekly periods, and used the average load of each time interval in the first two weeks to predict the third week. Figure 13 shows that this average is a good predictor, as errors in both experiments are low with root mean squared error (RMSE) of about 25. This means our predictions are 7-8% off from the actual load. This is a tolerable error that can be accounted for by adding a small safety margin and slightly over-provisioning the servers, as we did in our experiments. Note that the Second Life data shows a combination of natural user utilization cycles and scheduled jobs—the late-night peaks are due to a pool of 27 database machines performing snapshot operations. This shows that prediction works any time of workload that repeats over time. This indicates that, even for front facing applications, it is not uncommon for the database machines to experience relatively stable workloads. For such applications, Kairos consolidation strategies are likely to remain valid for long.

8. RELATED WORK

Significant research has been devoted to improve multi-tenancy and support for mixed-workloads inside a single DBMS [5, 11]. These advances allows us to rely on the design of modern DBMSs to deal with multiple workloads/tenants. In our experiments MySQL and PostgreSQL divided resources fairly among multiple databases, as long as the DBMS had sufficient hardware resources.

Database consolidation using virtual machines has been discussed as a way to reduce operational costs [17]. Soror et al. look at configuring multiple VMs running on the same hardware to support multiple database workloads [20]. Their work is suited to consolidating small number of workloads rather than the large numbers we consider, or when other requirements (e.g., security) call for a VM-based approach. Similarly, Soundararajan et al. presented a system for determining how to partition VM resources between multiple virtual machines on a single physical machine [21]. Their work is complementary to ours, as it could be used to optimize the assignment that is found by our system.

There have been several efforts to provide extreme levels of multi-tenancy [15, 3, 14], aiming to consolidate tens of thousands of nearly inactive databases onto a single server, especially when those databases have identical or similar schemas. The key challenge of this prior work has been on overcoming DBMSs' limitations at dealing with extremely large numbers of tables and/or columns. This is complementary to our work, as our approach can run into similar internal DBMS limitations. However, our workloads do not typically share any data or schemas.

Another relevant area of research is dynamic resource allocation in data centers [6, 4, 24]. Such work shares some of our goals, but is aimed at consolidating unpredictable stateless web workloads, with a focus on quick response to load changes and latency guarantees. This work combines sophisticated predictive models and dynamic reactive components, but ignores the issues that arise when migrating and consolidating data-intensive services like databases.

Gulati et al. analyze disk I/O for several types of workloads, presenting results on combining random and sequential I/O patterns that are consistent with our observations [9]. They also present a disk model that is not database specific, although more complex than ours [8]. Their evaluation only tests synthetic workloads. Ozmen et al. present a model that predicts the disk I/O created by a given database workload, without needing to measure the produc-

tion system [19]. Such a system is complementary to our work, as it reduces the need for detailed runtime statistics. There has been significant work on modeling of disk performance, usually from a non-database specific perspective. For example, Lee and Katz [18] present an analytical model for estimating the throughput of a combined load (consisting of a mixture of sequential and random I/Os) on a disk array; Varki et al [25] address a similar problem but build a model that includes caching and other effects.

Our use of a growing database table to put pressure on the buffer pool is somewhat similar VMware's "balloon driver" [26], which add memory pressure to force a guest OS to free unused or infrequently used pages. VMWare, however, doesn't use this technique to estimate RAM usage, but rather to force the OS to evict pages to make room for a new virtual machine. Furthermore, this the balloon driver runs inside of a guest OS, rather than inside of the DBMS. Such an OS-level technique is unlikely to work well, since the OS has no good way to choose which buffer pool pages to swap to disk, and may often choose pages containing data actively used.

Finally in the area of distributed database systems, substantial research has looked at the problem of allocating data to servers to optimize response-time or throughput [2, 22]. This work has different goals, since it aims at obtaining performance and scalability from a fixed set of machines, rather than consolidating them into the minimal number of physical nodes while meeting user performance expectations. Similarly, Aboulnaga et al. tackled the problem of "packing" map-reduce tasks on a cluster of machines, but the batch-oriented nature of the problem makes it significantly different from our scenario, leading to scheduling-oriented solutions [1].

Existing commercial systems [13] provide mechanisms for consolidation, but leave to the administrator the costly and error-prone task of devising a consolidation strategy. Our work automates the decision process, enabling larger scale and better consolidation.

9. CONCLUSION

In this paper, we presented a consolidation scheme for over-provisioned database servers; our approach allows enterprises to dramatically reduce the number of physical servers required to run their database infrastructure. We developed new monitoring techniques and resource models, as well as a non-linear mixed-integer programming approach to workload allocation, that analyze a database deployment and consolidate it. Our results, which show consolidation factors between 5.5:1 and 17:1, are based on experiments with synthetic and real-world data-sets from production servers running at MIT CSAIL, Wikia.com, Wikipedia, and Second Life. In addition to achieving high consolidation factors, our algorithms balance load across servers without affecting the throughput of the consolidated workloads. Additionally, we show that existing virtual machine technologies are not nearly as effective as our techniques at consolidating database workloads.

10. ACKNOWLEDGEMENTS

We thank A. Bergman of Wikia.net, D. Mituzas of Wikimedia.org, B. O'Connor of Linden Labs, and the The Infrastructure Group (TIG) at MIT CSAIL for providing the data sets used in the paper. This work was supported in part by Quanta Computer as a part of the T-Party Project.

11. REFERENCES

[1] A. Aboulnaga, Z. Wang, and Z. Y. Zhang. Packing the most onto your cloud. In *CloudDB*, 2009.

[2] P. Apers. Data allocation in distributed database systems. *ACM Transactions on Database Systems (TODS)*, 13(3):263–304, 1988.

[3] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. Multi-tenant databases for software as a service: schema-mapping techniques. In *SIGMOD*, 2008.

[4] M. Bennani and D. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *ICAC*, 2005.

[5] K. Brown, M. Carey, D. DeWitt, M. Mehta, and J. Naughton. Resource allocation and scheduling for mixed database workloads. Technical Report TR1095, University of Wisconsin - Madison CS Department, July 1992.

[6] A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *IWQoS*, 2003.

[7] C. Curino, E. P. C. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relationalcloud: a database service for the cloud. In *CIDR*, 2011.

[8] A. Gulati, C. Kumar, and I. Ahmad. Modeling workloads and devices for IO load balancing in virtualized environments. *SIGMETRICS Perform. Eval. Rev.*, 37(3):61–66, 2009.

[9] A. Gulati, C. Kumar, and I. Ahmad. Storage workload characterization and consolidation in virtualized environments. In *VPACT*, 2009.

[10] S. Harizopoulos, M. Shah, J. Meza, and P. Ranganathan. Energy efficiency: The new holy grail of data management systems research. In *CIDR*, pages 4–7, 2009.

[11] M. Heaton. Hosting Nirvana—The Future of Shared Hosting! [Online] `http://mattheaton.com/?p=185`, April 2009.

[12] K. Holmström. The TOMLAB optimization environment in Matlab. *Advanced Modeling and Optimization*, 1(1):47–69, 1999.

[13] HP. Polyserve: Product Overview. [Online] `http://bit.ly/azE3kA`, February 2009.

[14] M. Hui, D. Jiang, G. Li, and Y. Zhou. Supporting database applications as a service. In *ICDE*, pages 832–843, 2009.

[15] D. Jacobs and S. Aulbach. Ruminations on multi-tenant databases. *BTW Proceedings*, 2007.

[16] D. R. Jones. DIRECT global optimization algorithm. In *Encyclopedia of Optimization*, pages 725–735. 2009.

[17] D. Jonker. Combining database clustering and virtualization to consolidate mission-critical servers. Jan. 2009.

[18] E. K. Lee and R. H. Katz. An analytic performance model of disk arrays. *SIGMETRICS*, 21(1):98–109, 1993.

[19] O. Ozmen, K. Salem, M. Uysal, and H. S. Attar. Storage workload estimation for database management systems. In *SIGMOD*, 2007.

[20] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. *ACM Trans. Database Syst.*, 35(1), 2010.

[21] G. Soundararajan, D. Lupei, S. Ghanbari, A. D. Popescu, J. Chen, and C. Amza. Dynamic resource allocation for database servers running on virtual storage. In *FAST*, 2009.

[22] T. Stöhr, H. Martens, and E. Rahm. Multi-dimensional database allocation for parallel data warehouses. In *VLDB*, 2000.

[23] G. Urdaneta, G. Pierre, and M. van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11), 2009.

[24] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Trans. Auton. Adapt. Syst.*, 3(1), 2008.

[25] E. Varki, A. Merchant, J. Xu, and X. Qiu. Issues and challenges in the performance analysis of real disk arrays. *IEEE TPDS*, 15(6):559–574, 2004.

[26] C. A. Waldspurger. Memory resource management in VMware ESX server. In *OSDI'02*, pages 181–194, 2002.