LabFlow-1: a Database Benchmark for High-Throughput Workflow Management*

Anthony Bonner[1] bonner@db.toronto.edu

Adel Shrufi[1] shrufi@db.toronto.edu

Steve Rozen[2] steve@genome.wi.mit.edu

[1] University of Toronto, Department of Computer Science, Toronto, ON, Canada
[2] Whitehead/MIT, Center for Genome Research, Cambridge, MA, USA

Abstract. Workflow management is a ubiquitous task faced by many organizations, and entails the coordination of various activities. This coordination is increasingly carried out by software systems called *workflow management systems* (WFMS). An important component of many WFMSs is a DBMS for keeping track of workflow activity. This DBMS maintains an audit trail, or event history, that records the results of each activity. Like other data, the event history can be indexed and queried, and views can be defined on top of it. In addition, a WFMS must accommodate frequent workflow changes, which result from a rapidly evolving business environment. Since the database schema depends on the workflow, the DBMS must also support dynamic schema evolution. These requirements are especially challenging in high-throughput WFMSs—*i.e.*, systems for managing high-volume, mission-critical workflows. Unfortunately, existing database benchmarks do not capture the combination of flexibility and performance required by these systems. To address this issue, we have developed *LabFlow-1*, the first version of a benchmark that concisely captures the DBMS requirements of high-throughput WFMSs. LabFlow-1 is based on the data and workflow management needs of a large genome-mapping laboratory, and reflects their real-world experience. In addition, we use LabFlow-1 to test the usability and performance of two object storage managers. These tests revealed substantial differences between these two systems, and highlighted the critical importance of being able to control locality of reference to persistent data.

Appears in *Proceedings of the Fifth International Conference on Extending Database Technology (EDBT)*, March 25–29, 1996, Avignon, France, pages 463–478. Springer-Verlag, Lecture Notes in Computer Science, volume 1057.

* This work was supported by funds from the U.S. National Institutes of Health, National Center for Human Genome Research, grant number P50 HG00098, and from the U.S. Department of Energy under contract DE-FG02-95ER62101.

1 Introduction

1.1 Overview

Workflow management is a ubiquitous task faced by many organizations in a wide range of industries, from banking and insurance, to telecommunications and manufacturing, to pharmaceuticals and health care (*e.g.*, [8, 14]). The task is to coordinate the various activities involved in running an enterprise. Increasingly, this coordination is carried out by a software system called a *workflow management system* (WFMS). The activities themselves may use a variety of software components, including files, databases, application programs, and legacy systems, which may run on a variety of hardware platforms and operating systems, which may be located at a variety of sites. For example, in a large genome laboratory, workflow management software knits together a complex web of manual and automated laboratory activities, including experiment scheduling and setup, robot control, raw-data capture, multiple stages of preliminary analysis and quality control, and release of finished results. Appropriate software is necessary to make coordination of these activities both intellectually manageable and operationally efficient, and is a prerequisite for high-throughput laboratories. This software includes a DBMS component for tracking and controlling workflow activity. This paper addresses the requirements of this DBMS.

LabFlow-1 is a benchmark that concisely describes the database requirements of a WFMS in a high-throughput genome laboratory. Although it is based on genome-laboratory workflow, we believe that LabFlow-1 captures the database requirements of a common class workflow management applications: those that require a *production workflow system* [12]. In production workflow, activities are organized into a kind of production line, involving a mix of human and computer activities. Examples in business include insurance-claim or loan-application processing. Production workflow systems are typically complex, high-volume, and central to the organizations that rely on them; certainly these characteristics apply to the laboratory workflow-management systems used in high-throughput genome laboratories. Many production workflows are organized around central materials of some kind, which the workflow activities operate on. Examples of central materials include insurance claims, loan applications, and laboratory samples. As a central material is processed, workflow activities gather information about it.

Production workflow systems include the class of *Laboratory Information Management Systems*, or LIMS (*e.g.*, [14]). LIMS are found in analytical laboratories in a wide range of industries, including pharmaceuticals, health care, environmental monitoring, food and drug testing, and water and soil management. In all cases, the laboratory receives a continual stream of samples, each of which is subjected to a battery of tests and analyses. Workflow management is needed to maintain throughput and control quality.

Much of the research on workflow management in computer science has focussed on developing extended transaction models for specifying dependencies between workflow activities, especially in a heterogeneous environment (*e.g.*, [11, 8]). However, the *performance* of WFMSs has so far received little attention. The need to study performance arises because commercial products cannot support applications with high-throughput workflows. As stated in [8],

> Commercial workflow management systems typically support no more than a few hundred workflows a day. Some processes require handling a larger number of workflows; perhaps a number comparable to the number of transactions TP systems are capable of handling. For example, telecommunications companies currently need to process ten thousand service provisioning workflows a day, including a few thousand service provisioning workflows per hour at peak hours. Commercial workflow management systems are currently not capable of handling such workloads.

Note that each workflow may involve many transactions.

High-throughput workflows are also characteristic of large genome laboratories, like the Whitehead Institute/MIT Center for Genome Research (hereafter called "the Genome Center"). Workflow management is needed to support the Genome Center's large-scale genome-mapping projects [7]. Because of automation in instrumentation, data capture and workflow management, transaction rates at the Genome Center have increased dramatically in the last three years, from processing under 1,000 queries and updates per day in 1992 [9], to over 15,000 on many days in 1995. Of course, peak rates can be much higher, with a rate of 22.5 updates and queries per second recently observed over a 5-minute period. These rates are expected to increase by another order of magnitude in the near future if the Genome Center begins large scale sequencing of the Human genome [7]. Moreover, unlike the simple banking debit/credit transactions of some TPC benchmarks [17], these transactions involve complex queries, and updates to complex objects, such as arrays, sequences, and nested sets.

In this paper, we take a first step towards measuring the performance of workflow management systems. We do not attempt to study all the components that affect WFMS performance, such as networks, hardware platforms, and operating systems. Instead, we focus on one dimension of the problem: the performance of DBMSs that control and track workflow. Like other components, the DBMS can become a workflow bottleneck, especially in high-throughput applications.

1.2 DBMS Requirements

Workflow management has numerous DBMS requirements. First, it requires standard database features, such as concurrency control, crash recovery, consistency maintenance, a high-level query language, and query optimization. Workflow for advanced applications (such as laboratory workflow) also requires complex data types, as are found in object-oriented databases. A DBMS supporting production workflow management must provide this functionality on a mixed workload of queries and updates. In addition, it must provide two other features, which are characteristic of production workflow systems:

Event Histories. The DBMS must maintain an audit trail, or event history, of all workflow activity. From this history, the DBMS must be able to quickly retrieve information about any material or activity, for day-to-day operations. The history is also used to explore the cause of unexpected workflow results, to generate reports on workflow activity, and to discover workflow bottlenecks during process re-engineering. The DBMS must therefore support queries and views on an historical database. We note that many commercial laboratories are legally bound to record event histories. Salient examples include clinical drug trials and environmental testing.

Dynamic Schema Evolution. A hallmark of modern workflow management is that workflows change frequently, in response to rapidly changing business needs and circumstances [8]. Typically, a workflow will acquire new activities and existing activities will evolve. In both cases, the changed workflow generates new kinds of information, which must be recorded in the database. This requires changes to the database schema, preferably while the workflow is in operation (so-called *dynamic workflow modification*).

It is worth observing that because the database is historical and the schema is evolving, data at different points in the history will be stored under different schemas. Thus, an historical query or view may access data with many different schemas. This presents a challenge both to database design and to the formulation of queries and views. For instance, an application program may have to query an object's schema, as well as its value.

Existing database benchmarks do not capture the above requirements. This should not be surprising, as it has been observed by researchers working on OODBMS benchmarks that advanced applications are too complex and diverse to be captured by a single benchmark [4, 6]. A quick glance at several recent benchmarks illustrates their diversity of characteristics and requirements. For instance, the OO1, OO7 and HyperModel benchmarks [5, 4, 1] are concerned with the traversal of large graphs, which is a requirement of engineering and hypertext applications. In contrast, the SEQUOIA 2000 benchmark [19] is concerned with the manipulation of large sets of spatial and image data, such as those found in geographic information systems (GIS). The Set Query benchmark [15] is concerned with queries for decision support, including aggregation, multiple joins and report generation. (Such queries also arise in workflow management—for process re-engineering—but they are only part of the story.) Like these benchmarks, LabFlow-1 specifically targets a broad application area: workflow management. This application is characterized by a demand for *flexible* management of a stream of queries and updates, and of historical data and schema.

Unfortunately, many DBMSs do not yet fully support the requirements of production workflow systems as described above. Certainly, a surprising number of commercial products showed serious flaws in simple tests performed by the Genome Center in 1991. Fortunately, one can build a specialized DBMS that supports workflow on top of a storage manager that does not. This approach is taken at the Genome Center. Their specialized DBMS—called *LabBase* [16]—provides the needed support for event histories and schema evolution on top of an object storage manager. LabBase provides a historical query language, as well as structures for rapid access into history lists. It also transforms data from the user's database schema (which is dynamic) into the storage manager's schema (which is static). Besides providing support for workflow, this approach also provides portability, since different object storage managers can be "plugged into" the DBMS. In this way, we can test a wide range of existing storage managers. The benchmark implementation in this paper is based on this idea. We emphasize, however, that the LabFlow-1 benchmark does not depend on LabBase, which is an implementation detail. In the future, we hope to use our benchmark to test the support for workflow management in "off-the-shelf" DBMSs.

1.3 LabFlow-1

The LabFlow-1 benchmark is based on the data and workflow management needs of the Genome Center, and reflects their real-world experience. The benchmark has several goals. One goal is to provide a tool for the Genome Center to use in analyzing object storage managers for LabBase. Other goals are (*i*) to provide a general benchmark for databases supporting workflow-management systems, and (*ii*) to provide developers of next-generation DBMS technology with a set of requirements based on the characteristics of a real-world application domain. Developers of new technology often have few if any realistic applications against which to measure their systems. As with any benchmark, the challenge in designing LabFlow-1 was to create a database schema and workload that are realistic enough to be representative of the target class of applications yet simple enough to be feasibly implemented on a variety of systems.

Although LabFlow-1 is intended to be a general benchmark for DBMSs, this paper uses it to compare storage managers only. This is achieved by running the benchmark on versions of LabBase implemented on top of different storage managers, as described above. This paper compares ObjectStore (version 3.0) [13]. and Texas (versions 0.4 and 0.3) [18]. Compared to relational systems, these storage managers have been used in few production applications, so this analysis is interesting in its own right. Since they are a relatively novel technology, we compare these storage managers not only in terms of performance, but also in terms of client interface, tuning options, and system-administration facilities. In a similar fashion, LabFlow-1 can be used to compare other DBMS components, such as query optimizers and historical access structures.

Labflow-1 is a preliminary version intended for a single database user (hence the name). This choice arises from the architecture of the Genome Center's production DBMS, and also from a belief that it is useful to understand single-user performance before attempting to understand multi-user performance. We plan to develop in the future (*i*) multi-user and client/server workflow benchmarks, and (*ii*) benchmarks to evaluate and compare deductive database systems as query processors for workflow databases.

In sum, LabFlow-1 is the first version of a benchmark for DBMSs that control and track workflow. It is designed for applications with the following characteristics and requirements: high volume, mission critical workflows; frequent workflow change and process re-engineering; an audit trail of workflow activity; and complex-structured data. Due to space limitations, this paper can provide only an informal description of the benchmark. A detailed description can be found in [2]. Benchmark software is available at the following web site:

`ftp://db.toronto.edu/pub/bonner/papers/workflow/software/`

2 Workflow in LabFlow-1

As noted above, an important component of many workflow management systems is a DBMS for tracking workflow. This DBMS maintains an audit trail, or event history, that records the results of each activity. Like other data, the event history can be indexed and queried, and views can be defined on top of it. In genome laboratories, the event history serves much the same function as a laboratory notebook: it records what was done, when it was done, who did it, and what the results were. All laboratory information management systems produce an event history. Managing this event history is the primary function of the DBMS, and the main subject of this paper. This section provides an overview of data and workflow management in LabFlow-1 from the perspective of the DBMS. To keep the discussion concrete, we frame it in terms of laboratory workflow and LabBase.

The database has two main kinds of object: *materials* and *steps*. In object-oriented terms, the database has a material class and a step class, with subclasses representing different kinds of materials and steps. Step and material instances (objects) are created in two distinct situations. (*i*) As the laboratory receives or creates new materials, new material instances are created in the database to represent them. (*ii*) Each time a material instance is processed by a workflow step, a new step instance is created in the database to represent the activity. The step instance records the results of the workflow step (*e.g.*, measurements of length and sequence) and the conditions under which the step was carried out (*e.g.*, temperature and salinity). In this paper, we often say "step" instead of "step instance," and "material" instead of "material instance."

As a material is processed by a workflow, more-and-more step instances are created in the database, all associated with the same material. These steps constitute the material's event history. Workflow steps are not always successful, however, and earlier steps sometimes have to be repeated. Thus, the event history of a material may contain several different (though similar) versions of a step. Usually, the most recent version is of greatest interest to scientists, since it represents the most up-to-date results. LabBase uses special-purpose storage and access structures to rapidly retrieve most-recent results from the event history of each material [2].

Notice that as described above, the database schema depends on the workflow. For each kind of workflow step, there is a class in the database, and for each measurement made by the step, the class has an attribute. Consequently, workflow changes are reflected in the database as schema changes. In particular, as laboratory steps change, the corresponding classes also change. For instance, if scientists decide that a particular step should measure more properties, then attributes must be added to the corresponding class in the database. Likewise, if scientists add a new step to the workflow, then a new class is added to the database schema. If scientists split a complex step into a combination of smaller steps, then new classes are introduced into the schema to represent the smaller steps.

The data representation described above is *event oriented*. That is, information about a step is kept in one place, but information about a material is scattered among different steps. This provides a straightforward record of laboratory activity, but an awkward representation of materials. In particular, retrieving information about a material requires a detailed knowledge of laboratory workflow. For each property of a material, one must know what step(s) measured its value. Moreover, because workflows change constantly, a detailed knowledge of workflow changes and workflow history are also needed.

To alleviate these problems, the database provides a view that is *material oriented*, *i.e.*, in which information is associated with materials, not steps. In the database, observed values (*e.g.*, sequence, length, and mass) are attributes of *steps*; while in the view, they are attributes of *materials*. Using the view, an application programmer can retrieve the length of a DNA segment without knowing what step measured the length. In this way, the view isolates the application programmer from the details of workflow and workflow change.

Defining this view is not a straightforward matter. For one thing, the view definition depends on the workflow and its history. For another, different instances of the *same* kind of material can have *different* attributes in the view. For instance, segments of DNA may or may not have a `length` attribute, depending on whether or not they were processed by a step that measured their length. Thus, the attributes (or type) of a material depend on the *history* of the material, as well as its class. This rather dynamic feature reflects the flexibility demanded by workflow management. These issues are addressed in [2], where we introduce structures that allow the view to be defined *independently* of the workflow, so that the view definition does not have to be changed each time the workflow changes.

3 Benchmark Database Schema

Like the OO1 benchmark [5] and the Sequoia 2000 benchmark [19], our benchmark is independent of the data model provided by the DBMS. We therefore describe the database schema abstractly, using an extended entity relationship (EER) diagram [20]. An EER diagram is an ER diagram extended with `is-a` links. The database itself can be implemented in any number of ways, possibly as a relational database or as an object-oriented database.

As shown in Figure 1, the EER diagram has two levels (separated by a dashed line). The top level is abstract, and is a partial specification of an event-history model [2]. The second level is concrete, and represents the workflow in particular laboratory projects. The top-level is the same for all laboratory projects, while the bottom level changes from project to project. In fact, because of schema evolution, the bottom level also changes during the lifetime of a single project, possibly many times.

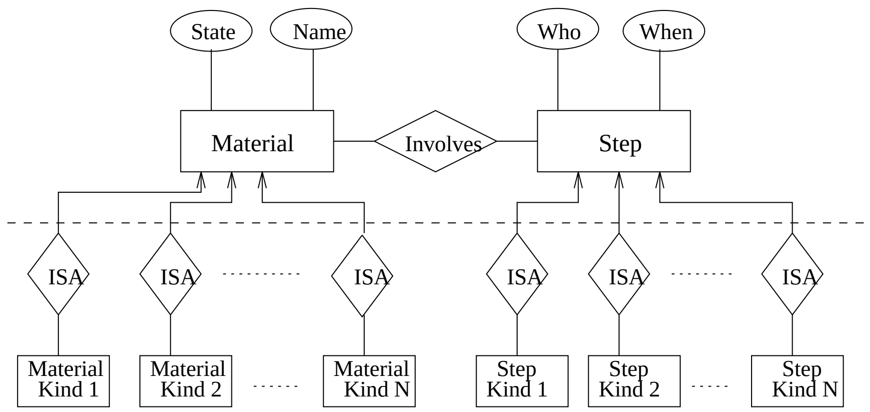

Fig. 1. An EER diagram for the benchmark schema

In the laboratory, materials are processed by experimental steps. The top level of the EER diagram thus has two entities: `material` and `step`. `Material` has two attributes: `name`, a human-recognizable name; and `state`, a workflow state. `Step` has two attributes: `who`, the name of the user (or program) that carried out the step; and `when`, the time and date at which the step was completed. Because of the `when` attribute, `step` data is historical. Each laboratory step involves at least one material, and each material is processed by at least one step. This is represented by the binary relation `involves`. Most queries in our benchmark require navigating between steps and materials via the `involves` relationship. The frequency of such queries arises because the database supports two mutually-dependent views of the data—a step-centered view, and a material-centered view—as described in Section 2. The `involves` relation allows data to be conveniently translated from one view to the other.

Different laboratory projects will involve different kinds of materials and different kinds of experimental steps. These appear as entities in the lower level of the ERR diagram. The two levels of the diagram are connected by `is-a` links. In object-oriented terms, each material kind is a subclass of `material` and inherits its two attributes, and each step kind is a subclass of `step` and inherits its two attributes. Each kind of material and step may also define its own attributes. The lower level of the EER diagram can be fleshed out in many different ways depending on the particular workflows being modeled. The benchmark provides a specific schema for this lower level [2]. It is a simplified version of schema used in production at the Genome Center.

In the rest of this paper, we adopt object-oriented terminology; we refer to entities as objects, where each object has a unique identifier; we refer to step kinds and material kinds as classes; and we say that objects are instances of classes.

4 Benchmark Database and Workload

The database for our benchmark is an event history or audit trail. We therefore need to provide a simple but realistic sequence of events, both to build the database and to serve as a workload for the benchmark. The issue of generating a realistic workload cannot be solved by generating a random sequence of simple events, as in TPC benchmarks. This would not capture the idea of materials flowing through a network of workflow stations. One solution would be to use a real database and a real workload from the Genome Center, generated during its day-to-day operations. We decided against this option because the resulting benchmark would be complex, hard to understand, and difficult to scale. Instead, our benchmark is based on a relatively simple workflow simulation [2]. This simulation models many of the important features encountered in real workflows at the Genome Center. For instance, the workflow has cycles, it has success and failure states, and it models multiple, co-operating production lines. As materials are processed by the workflow, each workflow activity produces a stream of database accesses, consisting largely of queries and inserts. These accesses are recorded as they arrive at the database, producing a *log* of database activity. Our benchmark is based on this log.

Starting from an empty database, the log produce a constantly expanding database. Our benchmark monitors the performance of the DBMS as the database expands. Specifically, we perform measurements at three different database sizes: when the database is half the size of physical main memory, the same size, and twice the size. By defining measurement points in terms of memory size, the benchmark can be scaled up as memories grow. It is not hard to extend the benchmark to larger multiples of main memory, and we plan to do so in the near future. However, since this significantly increases the time required to run benchmark tests, we intentionally limited database size in this initial study.

At each database size, we identify an interval of 10,000 queries and updates from the database log. We augment each interval with a small number of report-generation queries, to simulate the contribution to the DBMS workload by administrative departments. The three augmented intervals then become three workloads, one for each database size. The derivation of these workloads from the database log is illustrated in Figure 2.

To generate the database log, we ran our workflow simulation against a version of LabBase that relies on ObjectStore for storage management. We allowed the simulation to continue until the file size of the database was slightly larger than 64Mbyte. We refer to the log resulting from this simulation run as the *initial log*. We then modified the initial log as follows:

- Based on periodic measurements of the size of the database file during the simulation run, we selected points within the log that correspond to database-file

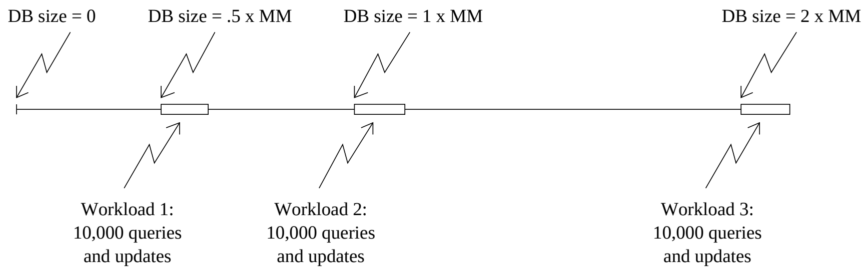

Fig. 2. The Derivation of Three Workloads from the Database Log

sizes of approximately 16Mbyte, 32Mbyte, and 64Mbyte. Since the benchmark machine has 32Mbyte of physical memory, these sizes are respectively 0.5, 1, and 2 times physical memory size.

• Starting at each of these three points in the log, we modified the subsequent sequence of 10,000 queries and updates. Specifically, we inserted queries to measure resource usage during execution of the log, and to measure resource usage for commits and for two read-only queries that simulate ad-hoc reporting activity. We also inserted queries to measure database size. We call each of the three modified sequences a *benchmark interval.*

• We removed all read-only queries that lay outside the benchmark intervals (in order to reduce the total amount of time needed to run the benchmark).

We call the result of the above modifications the *benchmark log.* The benchmark is run by executing LabBase with the benchmark log as input. Query output is saved in another file and compared between different benchmark runs to confirm consistent operation of the various versions of LabBase.

In more detail, the benchmark interval at each of the three database-file sizes consists of the following queries:

1. set baseline for resource usage
2. commit
3. get incremental resource usage
4. set baseline for resource usage
5. 10,000 queries and updates from the initial log
6. 2 report queries added to the log to simulate ad hoc queries
7. get incremental resource usage
8. set baseline for resource usage
9. commit
10. get incremental resource usage
11. get database size

In addition, at the start of the benchmark query stream, we inserted a single query to set the baseline resource usage. The benchmark thus records elapsed time, user cpu time, system cpu time, database size, and number of major page faults. All times are in seconds. Of these measurements, users will be most interested in elapsed time and database size. The other measurements are intended for database developers and debuggers.

5 Storage-Manager Comparisons

The purpose of our benchmark is to measure both the functional and the performance characteristics of DBMSs for managing workflow data. In the tests discussed here, we used several versions of the LabBase data server, which varied in storage management. The versions that we tested are:

1. OStore3—a version relying on ObjectStore (v3.0) [13] for storage management.
2. Texas.3—a version relying on the Texas storage manager (v0.3) [18] for storage management.
3. Texas.4—a version relying on the latest release (v0.4) of the Texas storage manager.
4. Texas.3+TC and Texas.4+TC— versions almost identical to Texas.3 and Texas.4 respectively, which use the same storage manager, but with additional object clustering implemented in client code.
5. Ostore3-mm, Texas.3-mm and Texas.4-mm—versions without any persistent storage management, and running entirely in main memory.

In a preliminary version of this paper we reported the results for Texas.3, but since then Texas.4 has been released. This new release employs a more efficient storage allocator that results in better data clustering and less internal fragmentation. In fact, the results for Texas.4, in terms of the elapsed time to run query 7, are comparable to those we obtained for Texas.3 with additional clustering, which clearly demonstrate the benefits of the new storage allocator. Although we conducted the same set of tests for both Texas.3 and Texas.4, some of the tests, such as Texas.4+TC and Texas.3-mm, gave similar results and did not lead to any new insights. Hence, they are not shown in Table 1. However, the complete set of test results can be found in [2]. Since much of the subsequent discussion applies to both Texas.3 and Texas.4, we often refer simply to Texas and its variants, Texas+TC and Texas-mm, without using version numbers.

The code running in the Ostore3-mm and Texas-mm tests was identical to that for the OStore3 and Texas tests, respectively, except that no persistent database was used. Technically, for the Ostore3-mm and Texas-mm tests all objects were allocated in the transient database. The consequence is that for Ostore3-mm, objects are allocated by `malloc`, whereas for Texas-mm the same allocator was used as for Texas and Texas+TC, since Texas uses its own version of `malloc` for both transient and persistent allocation.

ObjectStore and Texas present very similar interfaces to the programmer—their interface is essentially a persistent C++, in which C++ objects can be allocated in a persistent heap as well as in the familiar, transient heap. Both ObjectStore and Texas provide a distinguished mechanism for allocating objects on the persistent heap, but once an object is allocated all code that uses it can be oblivious of whether it is persistent or transient. The ObjectStore and Texas interfaces are so similar that out of approximately 11,000 lines of C++ code constituting the LabBase server, only a few hundred differ between the two storage managers. These differing sections are demarcated by precompiler directives (*e.g.*, `#ifdef/#endif` blocks.) Roughly speaking, the LabBase server implements the top level of the schema in Figure 1 (the part above the dashed line) directly in C++. Thus this part of the schema is compiled in. The lower level is implemented by making entries in a data dictionary, and is defined by database users at run time. The `involves` relation is implemented by keeping a (C++) pointer from each material instance to a list of pointers to those step instances that are related to the material by `involves`.

ObjectStore and Texas also both rely on pointer swizzling at page-fault time, as described in [18]. However, internally they are rather different. ObjectStore offers concurrent access with lock-based concurrency control implemented in a page server that mediates all access to the database.[3] Texas does not support concurrent access, and Texas programs access their database files directly.

Both the OStore3 and Ostore3-mm versions used ObjectStore indexes. All the Texas versions used b-tree indexes supplied by the authors of the Texas storage manager (though not part of the standard Texas distribution).

5.1 Performance Comparison

The benchmark machines for the results reported here were Sun 4/50s (SPARCstation IPXs) running SunOS 4.1.x. The systems had 32Mbyte of main memory. The benchmark machine was not used for any other purpose during benchmarking, but the benchmarks were not run in single-user mode. For Texas the database file, the executable, and the log were allocated on one disk, while the system swap space was allocated on another. Because of licensing restrictions and the undesirability of disturbing our production ObjectStore-based applications, we kept the ObjectStore database file, log file, and page server on a separate machine, a SPARCstation 20. The ObjectStore cache file (an `mmap`ed file used as temporary backing store by ObjectStore) was on the benchmark machine. We discuss the possible performance impact of these varying configurations later in this section. Table 1 summarizes the results for six different versions of the LabBase data server.

In each interval, query 7 measures resource usage for the 10,000 queries and updates from the initial log, plus the two additional queries that simulate ad hoc reporting queries. Query 11 measures database file size after a commit at the end of the benchmark interval.

At a nominal database size of approximately 0.5X main-memory size, all six versions of the LabBase data server consume similar resources, except for database-file size, which is greater in the Texas versions than in the OStore3

[3] Thus, when based on the ObjectStore storage manager, the LabBase server is what Carey et al. term a "client-level server" [3].

Intvl	Resource	Database Server Version					
		Persistent				Transient	
		OStore3	Texas.4	Texas.3+TC	Texas.3	Ostore3-mm	Texas.4-mm
0.5X	elapsed sec	1,424	1,474	1,469	1,402	1,384	1,471
	user cpu sec	1,381	1,452	1,449	1,385	1,364	1,450
	sys cpu sec	16	6	7	6	5	6
	majflt	329	571	468	397	463	683
	size (MB)	16.6	20.9	24.2	24.6	—	—
1X	elapsed sec	3,243	3,460	2,783	63,214	309,305	2,640
	user cpu sec	2,568	2,644	2,613	2,679	3,453	2,600
	sys cpu sec	137	83	15	2,958	20,981	8
	majflt	9,847	69,342	5,190	2,718,898	17,451,919	1,911
	size (MB)	33.8	41.5	48.2	48.8	—	—
2X	elapsed sec	7,057	n/a	n/a	n/a	n/a	n/a
	user cpu sec	4,711	n/a	n/a	n/a	n/a	n/a
	sys cpu sec	812	n/a	n/a	n/a	n/a	n/a
	majflt	153,742	n/a	n/a	n/a	n/a	n/a
	size (MB)	62.1	n/a	n/a	n/a	—	—

Table 1. Resource Usage for Six Storage-Management Schemes.

Elapsed time and user and system cpu were measured by SunOS's `times` *(3V) function (system-V flavor). "majflt" was measured by* `getrusage` *(2), and is the number of "page faults requiring physical I/O" (i.e., major faults). All resource usage was measured at query 7, except for the size of database files, which was determined using* `stat` *(2V) at query 11. The column labeled "intvl" indicates a particular benchmark interval at 0.5, 1, or 2 times physical memory size.*

version. (Of course, database file size is not meaningful for the Ostore3-mm or Texas.4-mm versions.) The larger size of the Texas database file is due to a segregated allocation scheme that rounds object sizes up to approximately the nearest power of 2, and which for the object sizes in LabBase left a substantial proportion of allocated space unused.

Starting at a nominal database size of approximately 1X main-memory size all the Texas versions began to thrash, and performance decayed so quickly that it seemed unfruitful to allow the benchmark run to continue to the 2X benchmark interval. The data-server process, as observed by SunOS's `ps` (1), was almost continually in 'D' state (non-interruptible wait—presumably for disk I/O). The larger database size in the Texas versions (than in OStore3) cannot be the primary cause of the problem, because the number of hard page faults (majflt) measured at query 7 for OStore3 at the even larger file size at the 2X interval was only 153,742, as opposed to 2,718,898 for Texas.3 at the 1X interval. Even in the case of the more efficient Texas.4, there is an 8-fold increase in page faults as compared to OStore3 at the 1X interval. In our use of ObjectStore in previous implementations of the production version of LabBase, we saw the beginnings of thrashing behavior, so we employed ObjectStore's ability to assign objects to storage segments (instances of the `os_segment` class) in an attempt to achieve better locality of reference than would be achieved in a single persistent heap. The OStore3 implementation of LabBase uses four such segments, three of which contain relatively small amounts of frequently accessed data, and one of which contains a relatively large amount of infrequently accessed data.

To test the hypothesis that poor locality of reference was the chief cause of thrashing behavior in Texas, we ran the same data server as in OStore3, but without any persistent database, and consequently without the benefit of clustering based on the `os_segment` class; we called this test Ostore3-mm. To our surprise Ostore3-mm displayed far worse thrashing behavior than all the Texas versions, and we deemed it unfruitful (and infeasible) to continue the benchmark to the 2X interval. Seeking to understand why page-faulting behavior in Ostore3-mm was so much worse than in Texas, we analyzed the object sizes in Texas.3, and hypothesized that Texas's segregated storage allocator, with one exception, stored frequently-accessed objects in segments different from those storing rarely accessed objects. In other words, by accident Texas's allocation provided reasonable storage segmentation, with the exception of one putatively mis-allocated class of objects.

To test this hypothesis, we modified the Texas.3 code to pre-allocate extents of the putatively mis-allocated objects. (Texas does not presently offer a storage-segment mechanism.) The Texas.3+TC test uses this modified Texas.3 code. The results in Table 1 show that, at the 1X benchmark interval, paging activity is indeed substantially reduced, which resulted in better performance than both Texas.3 and Texas.4. However, as the size of the database continued to grow beyond the 1X interval, paging activity increased dramatically, with the result that we did not gather performance numbers for the 2X interval of the Texas.3 test. We are still seeking an explanation for this behavior.

We also ran the same data server as Texas.4 without any persistent database, in the test called Texas.4-mm. Because Texas.4-mm still relies on its own storage allocator for the transient heap, and in the absence of any commit overhead, this test displays performance at the 1X interval that is slightly superior to all other tests.

None of the measurements presented in Table 1 captures commit costs. The ObjectStore page server resided on a different machine from the LabBase data server during the benchmark OStore3 tests, and it was not feasible for us to relate resource usage in the ObjectStore page server to activities of a particular LabBase client because the server was also being used for production activities. Therefore the only meaningful figure we can report for commits is elapsed seconds, presented in Table 2. We report the results for Texas.4 only since the commit times for Texas.3 were inordinately high due to an inherent inefficiency that has now been fixed in Texas.4. (Commit times for Ostore3-mm and Texas.4-mm were of course 0, and are not shown.) It is interesting to observe that Table 2 shows that the commit times for the two systems depend on the amount of work

Intvl	Query	OStore3	Texas.4
0.5X	3	47	109
	10	8	11
1X	3	70	115
	10	21	12
2X	3	162	n/a
	10	22	n/a

Table 2. Elapsed Time for Commits (in seconds).

committed. Clearly, in committing large amounts of work, OStore3 is more efficient than Texas.4 by almost a factor of two. Yet, Texas.4 performs as well as OStore3 for small amounts of work, and in fact outperforms it by the same factor at 1X main memory.

5.2 Functional Comparison

In contrast with commercial relational database management systems, there are huge differences in the functional and operational characteristics of today's object databases. Therefore a critical objective of our benchmark is to shed light on similarities and differences of such functional and operational characteristics. The interested reader is referred to [2] for a discussion of the similarities and differences between ObjectStore and the Texas Storage System. We compare the two systems along several dimensions: concurrency control, class libraries, tuning options, and database administration.

To summarize, our benchmark revealed significant differences among different storage managers and implementations of LabBase. The most notable difference between the storage managers was in facilities to control object allocation by storage segment, which critically affects performance of the benchmark application when the database size exceeds main-memory size. The benchmark also revealed significant differences in the time needed to perform commits in the two systems.

Acknowledgements: The authors gratefully acknowledge the support of the Whitehead Institute for Biomedical Research and the Whitehead/MIT Center for Genome Research for making this work possible. We also express our thanks to the developers of the Texas storage system.

References

1. T.L. Anderson, A.J. Berre, M. Mallison, H.H. Porter, and B. Schneider. The hypermodel benchmark. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 317–331, Venice, Italy, March 1990.

2. A. Bonner, A. Shrufi, and S. Rozen. LabFlow-1: a database benchmark for high-throughput workflow management. Technical report, Department of Computer Science, University of Toronto, 1995. 53 pages. Available at `http://www.db.toronto.edu:8020/people/bonner/bonner.html`.
3. M.J. Carey, D.J. DeWitt, M.J. Franklin, et al. Shoring up persistent applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 383–394, Minneapolis, MN, May 1994.
4. M.J. Carey, D.J. DeWitt, and J.F. Naughton. The OO7 benchmark. Technical report, Computer Sciences Department, University of Wisconsin-Madison, January 1994. Available at `ftp://ftp.cs.wisc.edu/oo7/techreport.ps`.
5. R.G.G. Cattell. An engineering database benchmark. In [10], chapter 6, pages 247–281.
6. A. Chaudhri. An Annotated Bibliography of Benchmarks for Object Databases. *SIGMOD Record*, 24(1):50–57, March 1995.
7. *Communications of the ACM*, 34(11), November 1991. Special issue on the Human Genome Project.
8. D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to infrastructure for automation. *Journal on Distributed and Parallel Database Systems*, 3(2):119–153, April 1995.
9. Nathan Goodman. An object oriented DBMS war story: Developing a genome mapping database in C++. In Won Kim, editor, *Modern Database Management: Object-Oriented and Multidatabase Technologies*. ACM Press, 1994.
10. Jim Gray, editor. *The Benchmark Handbook for Database and Transaction Processing Systems*. Morgan Kaufmann, San Mateo, CA, 1991.
11. M. Hsu, Ed. Special issue on workflow and extended transaction systems. *Bulletin of the Technical Committee on Data Engineering (IEEE Computer Society)*, 16(2), June 1993.
12. Setrag Khoshafian and Marek Buckiewicz. *Introduction to Groupware, Workflow, and Workgroup Computing*. John Wiley & Sons, Inc., 1995.
13. Charles Lamb, Gordon Landis, Jack Orenstein, and Dan Weinreb. The ObjectStore database system. *Communications of the ACM*, 34(10):50–63, October 1991.
14. Allen S. Nakagawa. *LIMS: Implementation and Management*. Royal Society of Chemistry, Thomas Granham House, The Science Park, Cambridge CB4 4WF, England, 1994.
15. P. O'Neal. The set query benchmark. In [10], chapter 5, pages 209–245.
16. Steve Rozen, Lincoln Stein, and Nathan Goodman. Constructing a domain-specific DBMS using a persistent object system. In M.P. Atkinson, V. Benzaken, and D. Maier, editors, *Persistent Object Systems*, Workshops in Computing. Springer-Verlag and British Computer Society, 1995. Presented at POS-VI, Sep. 1994. Available at `ftp://genome.wi.mit.edu/pub/papers/Y1994/labbase-design.ps.Z`.
17. O. Serlin. The history of debit credit and the TPC. In [10], chapter 2, pages 19–117.
18. Vivek Singhal, Sheetal V. Kakkad, and Paul R. Wilson. Texas: an efficient, portable persistent store. In *Proceedings of the Fifth International Workshop on Persistent Object Systems (POS-V)*, San Minato, Italy, September 1992. Available at `ftp://cs.utexas.edu/pub/garbage/texaspstore.ps`.
19. M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The Sequoia 2000 storage benchmark. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 2–11, Minneapolis, MN, May 1993.

20. T.J. Teorey, D. Yang, and J.P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18:197–222, June 1986.