

# Oasis: An Architecture for Simplified Data Management and Disconnected Operation

Anthony LaMarca<sup>1</sup>, Maya Rodrig<sup>2</sup>

<sup>1</sup>Intel Research Seattle

<sup>2</sup>Department of Computer Science & Engineering, University of Washington  
anthony.lamarca@intel.com rodrig@cs.washington.edu

**Abstract.** Oasis is an asymmetric peer-to-peer data management system tailored to the requirements of pervasive computing. Drawing upon applications from the literature, we motivate three high-level requirements: availability, manageability and programmability. Oasis addresses these requirements by employing a peer-to-peer network of weighted replicas and performing background self-tuning. In this paper we describe our architecture and an initial implementation. Our performance evaluation and implementation of three applications suggest that Oasis offers good availability and performance while providing a simple API and a familiar consistency model.

## 1 Introduction

The vision of pervasive computing is an environment in which users, computing and the physical environment are artfully blended to provide insitu interactions that increase productivity and quality of life. While many of the hardware components required to realize this vision are available today, there is a dearth of robust applications to run on these new platforms. We contend that there are so few pervasive computing applications because they are too hard to develop, deploy and manage. A number of factors that are particular to pervasive computing scenarios make application development challenging: devices are resource-challenged and faulty, and devices may be continually arriving and departing. While prototypes of compelling applications can be deployed in the lab, it is very difficult to build an implementation that is robust and responsive in a realistic pervasive environment.

We argue that the best way to foster pervasive computing development is to provide developers with a comprehensive set of software services, in effect an “OS for pervasive computing”. While there has been work in the area of system software for pervasive computing, a number of significant challenges remain [17]. In this paper, we address the challenge of providing pervasive computing support for one of the more traditional services, namely the storage and management of persistent data.

We examined fifteen pervasive computing applications described in the literature and we distilled a common set of requirements that fall in the areas of *availability*, *manageability*, and *programmability*. Based upon these requirements, we designed and implemented a data management system called Oasis. We tested the performance

of Oasis and used it to implement three pervasive computing applications in order to understand how well it satisfies these requirements.

The contributions of this work are twofold. First, we offer an investigation of the data management requirements of pervasive computing applications. Second, we propose a new architecture to address these requirements that combines existing systems and database techniques and algorithms in a new way.

The rest of the paper is organized as follows. In section 2 we identify the data management requirements of pervasive computing applications and draw specific examples from the literature. Section 3 presents the Oasis architecture and our initial implementation. In section 4 we describe our experience constructing three applications on top of Oasis. We discuss the performance of the system as measured with the workload of one of our applications in section 5. In sections 6, 7 and 8 we compare Oasis to related work, describe future work, and conclude.

## 2 Data Management Requirements of Pervasive Computing

Through a survey of fifteen pervasive computing applications published in the literature [1,3,5,6,13,14,16,20,21,22,26,28,31,32], we have identified what we believe are the important data management requirements of these applications. The breadth of applications covered in the survey includes smart home applications, applications for enhancing productivity in the workplace, and Personal Area Network (PAN) applications. The specific requirements can be grouped into three areas: *availability*, *programmability*, and *manageability*.

### 2.1 Availability

Pervasive computing applications are being developed for environments in which people expect devices to function 24 hours a day, 7 days a week. The AwareHome [16] and EasyLiving [6], for example, augment household appliances such as refrigerators, microwaves, and televisions that typically operate with extremely high reliability. For many of these augmented devices to function, uninterrupted access to a data repository is needed, thus a storage solution for pervasive computing must ensure data is available in the following conditions:

*Data access must be uninterrupted, even in the face of device disconnections and failures.* Proposed pervasive computing scenarios utilize existing devices in the home as part of the computing infrastructure [6,20]. A data management solution should be robust to the failure of some number of these devices; turning off a PC in a smart home should not cause the entire suite of pervasive computing applications to cease functioning. The data management system must handle both graceful disconnections and unexpected failures, and data must remain available as devices come and go.

*Data may need to be accessed simultaneously in multiple locations even in the presence of network partitions.* The majority of applications we examined include scenarios that require support for multiple devices accessing the same data in multiple

locations. Commonly, these applications call for users to carry small, mobile devices that replicate a portion of the user's home or work data [22,31]. Labscape [3] cites disconnection as uncommon, but would like to support biologists who choose to carry a laptop out of the lab. Finally, some applications involve non-mobile devices sharing data over unreliable channels. The picture frame in the Digital Family Portrait [26], for example, communicates with sensors in the home of a geographically remote family member. In all of these cases, the application scenarios assume the existence of a coherent data management system that supports disconnected operation.

*Data can be accessed from and stored on impoverished devices.* Pervasive computing applications commonly involve inexpensive, resource-constrained devices used for both accessing and storing data. In PAN applications, for example, impoverished mobile devices frequently play a central role in caching data and moving data between I/O and other computational devices [21,22,31]. Ideally a data management system for pervasive computing would accommodate the limitations of resource challenged devices; challenged devices would be able to act as clients, while data could be stored on fairly modest devices.

## **2.2 Manageability**

Perhaps the single largest factor keeping pervasive computing from becoming a mainstream reality is the complexity of managing the system. We have identified a number of features that are essential to making a data management system for pervasive computing practical for deployment with real users.

*Technical expertise should be required only in extreme cases.* By many accounts the "living room of the future" will have the computational complexity of today's server room; however there will rarely be an expert to manage it. In many cases only non-technical users are present [26] while in extreme applications like PlantCare [20], there are no users at all. In the spirit of IBM's autonomic computing initiative [34], data management for pervasive computing environments should be self-managing to the largest extent possible.

*Adjustments to storage capacity should be easy and incremental.* Many of the pervasive computing systems we examined could most appropriately be labeled as platforms on which many small applications and behaviors are installed [5,6,13]. In such an environment, the data management system should be able to grow incrementally to support changing workloads and capacity needs.

*The system should adapt to changes within and across applications.* The wide variety of devices and applications suggests that the data management system should monitor and adapt to changes in configuration and usage. Consider the location tracking system that is common to many pervasive computing scenarios [3,6,32]. Its job is to track people and objects and produce a mapping for other applications to use. In some scenarios this location data is used infrequently while other scenarios may require hundreds of queries against this data per second. A static configuration runs the risk of either providing poor performance or over-allocating resources. An adaptive

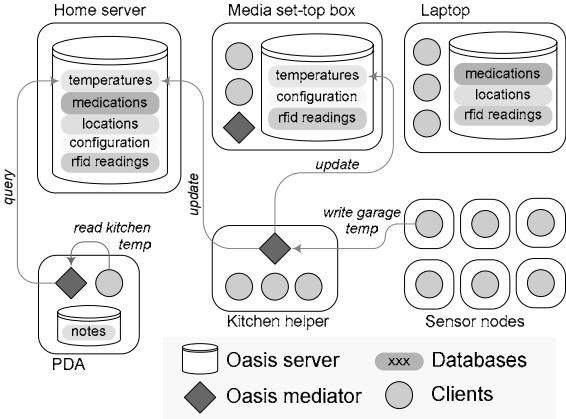
solution, on the other hand, could detect the activation of a demanding application and adjust priorities accordingly, ensuring good performance and overall system efficiency.

**2.3 Programmability**

A distributed, dynamic and fault-ridden pervasive computing environment is a far cry from the computing platforms on which most software engineers are trained. With this in mind, we have identified a number of requirements intended to lower the barrier to entry and make reliable, responsive pervasive computing applications easier to develop.

*The system should offer rich query facilities.* Pervasive computing applications often involve large amounts of structured sensor data and frequent searches through this data. A common pattern is that an application takes an action if a threshold value is crossed (e.g., going outdoors triggers a display change [26], low humidity triggers plant watering [20], proximity to a table triggers the migration of a user interface [3]). Data management systems that provide indexing and query facilities vastly reduce the overhead in creating efficient implementations of such behaviors.

*The system should offer a familiar consistency model.* Some distributed storage systems provide “update-anywhere” semantics [30] in which clients are permitted to read and write any replica at any time, even when disconnected. These systems provide weak consistency guarantees and applications may see writes to the data occur in a different order than they were written in. These weak guarantees can result in a wide range of unpredictable behaviors that make it difficult to write reliable applications. Our experience suggests that a familiar, conservative consistency model is more appropriate for most pervasive computing applications, even if it results in a decrease in availability.



**Fig. 2. A sample configuration of devices, databases and Oasis components**

*The system should provide a single global view of the data.* Some application scenarios dictate a specific replica configuration in order to achieve particular semantics. However, many applications merely want to reliably store and retrieve data. Accordingly, a data management system for pervasive computing should include a facility for automatic data placement and present the view of a single global storage space to application developers. While these decisions can be guided by hints given by the application, the developer should not be directly exposed to a disparate collection of storage devices.

### **3 The Oasis Architecture**

Oasis is a data management system tailored to the requirements of pervasive computing. In Oasis, clients access data via a mediator service that in turn communicates with a collection of Oasis servers. The mediator service stores no persistent data; its only purpose is to run the Oasis consistency protocol. (The mediator's function has been separated to allow the participation of impoverished clients like sensor beacons.) Figure 1 shows an example of an Oasis configuration in an instrumented home. Data is replicated across Oasis servers to provide high availability in the event of a device disconnection or failure. Oasis does not depend on any single server; data remains accessible to clients provided a single replica is available. In the remainder of this section we describe the Oasis architecture and describe how these enable Oasis to meet the requirements described in Section 2.

#### **3.1 Data Model**

From the client's perspective, Oasis is a database that supports the execution of SQL queries on relational data. We chose SQL because it is a widespread standard for accessing structured data. An Oasis installation stores a number of databases. Each database holds a number of tables that in turn hold a set of records. We envision that different databases would be created for different types of data such as sensor readings, configuration data, sound samples, etc. In Section 4, we describe three applications we have built using Oasis and their data representation. It should be noted that nothing in the rest of the architecture is specific to the relational data model, and Oasis could manage file- or tuple-oriented data with a few small changes.

#### **3.2 P2P Architecture with Replication**

As devices may arrive and depart in pervasive computing scenarios, an architecture that supports dynamic membership is needed. A pure peer-to-peer (P2P) architecture provides the desired decentralization, adaptability, and fault-tolerance by assigning all peers equal roles and responsibilities. However, the emphasis on equal resource contribution by all peers ignores differences in device capabilities. To support a wide

variety of devices, Oasis is an asymmetric-P2P system, or “super-peer” system [34], in which devices’ responsibilities are based on their capabilities. Devices with greater capabilities contribute more resources and can perform computation on behalf of others, while impoverished devices may have no resources to contribute and participate only as clients.

Data is replicated across multiple Oasis Servers to provide high availability. In our initial implementation, replication is done at the database level. (Replicating entire databases simplifies implementation but potentially overburdens small devices. In Section 7 we discuss the potential for partial replication.) An initial replica placement is determined at creation time and is then tuned as devices come and go, and data usage changes. The self tuning process is described in Section 3.4.

### 3.3 Weighted-Voting and Disconnection Operation

All distributed data-stores employ an access-coordination algorithm that offers a consistency guarantee to clients accessing the data. To provide developers with a familiar consistency model, we chose an algorithm for Oasis that offers clients *sequential consistency* [23] when local replicas are available. Sequential consistency guarantees that the operations of all clients execute in some sequential order, and the operations of each client appear in this total ordering in the same order specified by its program. Basically, sequential consistency provides a set of distributed clients with the illusion that they are all running on a single device.

The traditional way to provide sequential consistency and allow disconnected operation is with a quorum-based scheme in which a majority of the replicas must be present to update the data. We have adapted Gifford’s “weighted voting” [10] variant of the basic quorum scheme. As in a quorum-based scheme, data replicas are versioned to allow clients to determine which replica is the most recent. In addition, weighted voting assigns every replica of a data object a number of votes. The total number of votes assigned to all replicas of the object is  $N$ . A write request must lock a set of replicas whose votes sum to at least  $W$ , while read operations must contact a set of replicas whose votes sum to at least  $R$  votes. On a read, the client fetches the value from the replica with the highest version number. On a write, the client must update all of the replicas it has locked and then apply the new update. Weighted voting ensures sequential consistency by requiring that  $R+W>N$ . This constraint guarantees that no read can complete without seeing at least one replica updated by the last write (since  $R>N-W$ ). Weighted voting is more flexible than traditional quorum-based approaches because the vote allocation as well as  $R$  and  $W$  can be tailored to the expected workload. Making  $R$  small, for example, boosts performance by increasing read parallelism. Making  $R$  and  $W$  close to  $N/2$  allows up to half the servers to fail, increasing fault-tolerance.

Gifford’s original weighted-voting algorithm was written for a single, centralized client accessing data on a collection of servers. With a single client, the metadata for a replicated data object ( $R$ ,  $W$ ,  $N$  and the list of replica locations and votes) can be maintained in a centralized fashion at the client’s discretion. To allow weighted voting to operate in a P2P system with multiple clients and servers, we developed a

decentralized version of Gifford’s algorithm. In our scheme, versioned copies of the metadata are distributed along with the data in each replica. Updating the metadata requires acquiring a quorum of  $W$  votes, akin to a data update. This allows data and metadata operations to be safely interleaved, enabling the system to perform self tuning. To guarantee sequential consistency, we add the additional constraint that  $W \geq R$ . This ensures that both reads and writes of the data see the latest version of the metadata (since  $W \geq R > N - W$ ). For more detail about our decentralized weighted-voting algorithm see [29].

In order to ensure consistency, writes cannot proceed when the required number of votes is not available. (In Oasis, a database appears to be “read only” when insufficient votes are available.) Read queries, on the other hand, are permitted to proceed even if a quorum cannot be attained when a local replica is available. This is the equivalent of allowing a disconnected client to read from a stale cache of the data. While this may seem to violate sequential consistency, it does not. Since the client cannot acquire a read quorum, they also cannot write the data ( $W \geq R$ ), ensuring they see a consistent, if out-of-date, snapshot of the data. When Oasis performs a query on a potentially stale replica, the query results are marked as stale to alert the client.

Update requests can potentially fail if either the mediator or one of the replicas crashes or departs during the operation. To ensure the consistency of the database, mediators use a two-phase commit protocol when acquiring votes and executing updates on a replica. If a request fails to complete on a replica, the replica will be marked as invalid. Invalid replicas cannot participate in client operations until a distributed recovery algorithm [11] has been successfully executed.

### **3.4 Online Self-Tuning and Adaptability**

Oasis was designed to support self-tuning. The SQL data-model provides the opportunity to add and delete indices. Our weighted voting scheme permits flexibility regarding the number of placement of replicas, the vote allocation, and the values of  $R$  and  $W$ . Finally, our consistency scheme allows these parameters to be adjusted during a stream of client requests. This allows Oasis to be tuned in an online fashion without denying applications access to the data or requiring user intervention.

Applications have the choice of managing their own replica placement and vote assignment, or allowing Oasis to manage the data on their behalf. For applications that do not want to manage their own replica placement, Oasis includes a self-tuning service that automatically handles replica configuration. When databases are created in Oasis, performance and availability expectations can be provided by applications that want auto-tuning. Oasis servers advertise their performance and availability characteristics and the self-tuner uses these along with the application expectations to make its configuration decisions. The self-tuner periodically examines each database's expectations and checks if they are best served by their current replica placement and vote assignment, making adjustments if appropriate. As we discuss in Section 7, we see the development of more sophisticated self-tuning behaviors based on machine learning techniques as a promising direction to pursue for future research.

### 3.5 Implementation Details

Our initial implementation of Oasis was written in Java and our servers and mediators communicate using XML over HTTP. Oasis was implemented as a meta-database that delegates storage and indexing to an underlying database. The Oasis server has been written to run on top of any JDBC-compliant database that supports transactions. Our initial deployments have used a variety of products: PostgreSQL and MySQL have been used on PC-class devices, and PointBase, an embedded, small-footprint database, has been used with IPAQs and other ARM-based devices.

## 4 Applications

To investigate usability, we implemented three applications on top of Oasis. Two of these are variants of existing applications from the literature while Guide is a new application that has been developed in our laboratory. While we did not undertake a rigorous evaluation of our implementations, our experience suggests that Oasis is well suited for the pervasive computing domain. More interestingly, for all three applications, we encountered ways in which the capabilities of Oasis transparently augmented or extended some basic function of the application.

### 4.1 Portrait Display

The Portrait Display is an ongoing project in our laboratory motivated by Mynatt et al.'s Digital Family Portrait [26]. The Digital Family Portrait tries to increase the awareness of extended family members about an elderly relative living alone. Information about an elderly person (health, activity level, social interaction) is collected by sensors in his instrumented home, and unobtrusively displayed at the remote home of an extended family member on a digital picture frame surrounding his portrait. Researchers in our laboratory have been using the digital family portrait scenario to explore various approaches for displaying ambient data about elders that require home care. In conjunction with their investigation, we have implemented a digital portrait inspired by the original that runs on top of Oasis. The four categories of information displayed in our digital portrait are medication intake, meals eaten, outings, and visits. The data used to generate the display comes from a variety of sources. In our prototype, medication and meal information are gathered using Mote-based sensors [12] and cameras, while information about visits and outings is currently entered by hand using a web interface.

The relational data model provided by Oasis is well suited for describing the regular, structured data used by the portrait display application. Similarly, the types of queries needed to extract results to display are easily expressed in SQL.

Oasis effectively supports the availability requirements of the portrait display. The portrait display uses a separate Oasis database for each category of information collected (meals, visits, etc.). Each database is explicitly configured with a replica

with 4 votes that resides on the device where the data is gathered and a 1-vote replica on the portrait display device (N=5, R=3, W=3). This configuration allows the data to be read and updated at its source, and when a connection exists, allows the portrait display to obtain recent changes. Note that this remains true even if the data source itself is disconnected. For example, after visiting with the elder, a care provider can enter notes about the visit while sitting in his car or back at his office, a practice mentioned in our fellow researcher's interviews with care providers. While the ability to record information when disconnected was not part of the original scenario, the capability is provided by Oasis transparently by placing the 4-vote replica on the care providers laptop or PDA.

This configuration also supports unplanned disconnections by the portrait display itself. The original digital family portrait used a simple client-server model in which the display was rendered as a web page fetched from a server running in the elder's home. While suitable for a prototype, it would not work well in a real deployment in which DSL lines and modem connections do in fact go down at times. Implementations that rely on a web client-server model must either display an error page or leave the display unchanged in the case of a disconnection. With Oasis, disconnections are exposed in the form of stale query results giving the application the opportunity to display the uncertainty in an appropriate way.

#### **4.2 Dynamo: Smart Room File System**

Stanford's iRoom [13] and MIT's intelligent room [5] are examples of "productivity enhanced workspaces" in which pervasive computing helps a group of people work more efficiently. In their scenarios, people gather and exchange ideas while sharing and authoring files using a variety of viewing and authoring tools. Generally, in these scenarios either: 1. the files are stored on a machine in the workspace and users lose access when they leave the space, or 2. files reside on a user's personal device (like a laptop) and everyone else in the workspace loses access when the user departs.

For our second application we developed a system called Dynamo that allows file-oriented data to be consistently replicated across personal devices. In Dynamo, each user or group owns a hierarchical tree of directories and files, much like a home directory. Users can choose contexts in which to share various portions of their file system with other users (example contexts are 'code review' or 'hiring meeting'). The collective sum of files shared by the users that are present make up the files available in the workspace. In this manner, everyone present at a hiring meeting can share their proxies and interview notes with the other participants without exposing other parts of their file space.

Dynamo was written as an extension to Apache's webDav server that stores a user's files in an Oasis database. Microsoft's Web Folders are used to mount the webDav server as a file system, allowing Dynamo's file hierarchy to be accessed using standard desktop applications. Implementing Dynamo on top of Oasis required a small number of changes to the original webDav server (less than 400 lines). Despite this, the relational data model was not a good fit for the file oriented data

stored in Dynamo. Mapping the hierarchy of the file system into relations required a translation step not needed in our other two applications.

The flexibility of Oasis enabled a variety of semantically interesting configurations. If desired, Dynamo can create a 1-vote replica of a user's files on a device that resides in the workspace. This permits the user to disconnect and leave while enabling the remaining participants to view (but not write) the files that have been shared. These stale read-only files remain in that context until the user returns at which time a more up to date, writeable version would be seen. For files owned by a group, interesting ownership policies can be arranged by assigning votes based on the user's roles and responsibilities. This can be used to enforce policies ranging from basic majority schemes in which all replicas are equal, to more complex specifications such as: 'the budget cannot be edited unless the boss plus any other two employees are present'. While this flexibility raises a number of privacy and interface challenges, it shows how Oasis can add rich semantics to a simple application.

### 4.3 Guide

The Guide project [9] aims to use passive Radio Frequency Identification (RFID) tags, for the purpose of context inference. The project involves tagging thousands of objects in a work space with RFID tags and tracking their position using RF antennas mounted on a robot. Tagged objects include books, personal electronics and office/lab equipment. As the robot moves around the environment the antennas pick up the ID of nearby tags. For each tag ID  $i$  discovered at time  $t$  and location  $l$ , the platform writes the tuple  $(i, t, l)$  to a database. The database thus accumulates the location of objects over time. The goal of Guide is to determine high-level relationships between objects based on the accumulated data.

To help in our evaluation, the Guide team in our lab implemented their system on top of Oasis. The relational data model was an ideal match for Guide's structured RFID readings and the Guide workload could be easily expressed as SQL statements. The indexing provided by the underlying database was essential in reducing the time to process Guide queries.

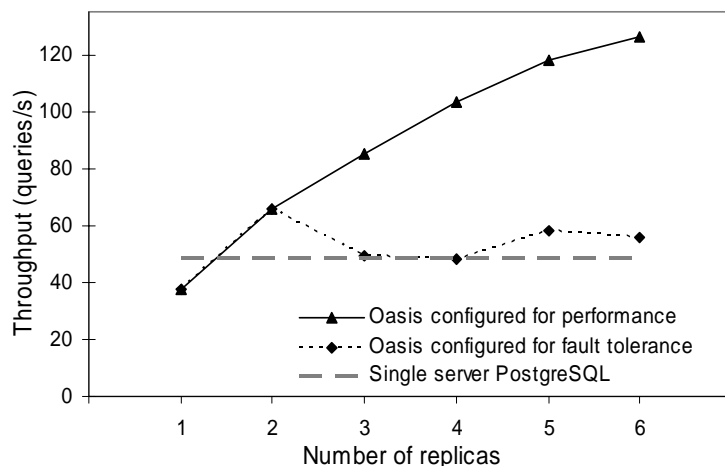
Guide has demanding performance, reliability and availability requirements. First it is expected to generate large quantities of data; the database is expected to grow to contain millions of readings in three months. Given that the Guide database is intended to be used as a common utility, it is quite possible that tens or hundreds of clients will query the guide database. The database must therefore scale to support large numbers of potentially complex queries in parallel. Second, this large quantity of data must be stored reliably. Since the data may represent months of activity (and it is impossible to regenerate the data), and the entire period may well be relevant, losing the data will be detrimental. Third, since the queries may be part of standing tasks (such as context-triggered notification) it is important that the database be highly available. Based on Guide's goals of high availability and performance, the Oasis self-tuner configured the Guide database with three 1-vote replicas ( $N=3$ ,  $R=2$ ,  $W=2$ ). This configuration provides high reliability, good performance and continuous access to the data provided that any two of the three servers are available.

## 5 Performance

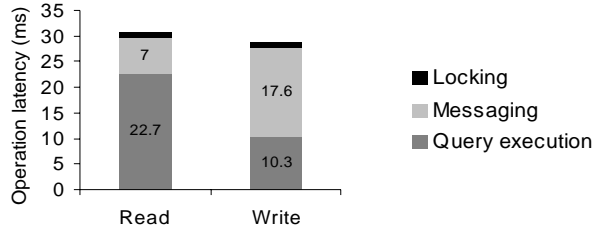
To measure the performance of Oasis using a realistic workload, we constructed an experiment based on the Guide application described in Section 4.3. The Guide database is comprised of 3 tables: a *reading* table tracking when and where an RFID-tag was seen, an *object* table that relates RFID-tags to object names (like ‘stapler’), and a *place* table that records the geometric bounds of rooms. Our experimental data set was seeded with 1 million records in the reading table, 1000 records in the object table and 25 records in the place table. This approximates the number of tagged objects in our laboratory and the number of readings we expect to record in a month.

In our benchmark a set of clients alternate between performing queries and updates on the database. The queries are all of the form “Where was object X last seen?”. These are fairly compute intensive queries that join across the reading and place tables. The updates are inserts of a new record into the reading table. The ratio of queries to updates performed by the clients is 50:1, again approximating the expected workload in a Guide deployment. To show the tradeoffs offered by Oasis, we measure two Oasis configurations: one which offers the highest query performance ( $R=1, W=N$ ) and another which offers the highest tolerance to server failure ( $R=N/2, W=N/2+1$ ). To show the overhead that Oasis introduces, we compare its performance to direct accesses to the underlying PostgreSQL database. In our experiments, the number of clients is fixed at 10 and the number of replicas is varied from 1 to 6. Each client and server in the test ran on its own Pentium 4 PC running Windows 2000 connected via 100MB/s Ethernet. The Oasis servers and clients ran on Sun’s 1.3.1 JVM and the underlying data was stored in PostgreSQL 7.3.

Figure 2 shows the total throughput achieved by the set of clients. The graph shows that for a singly-replicated database, Oasis achieves lower throughput than



**Fig. 3.** This graph compares the throughput of two Oasis configurations and a single PostgreSQL server. In the experiment ten clients are running the Guide workload concurrently



**Fig. 3. The latency breakdown of read and write queries in the Guide workload for Oasis configured with two replicas.**

PostgreSQL. This is expected since Oasis incurs additional overhead running our locking protocol. The graph shows that as replicas are added, read queries are able to take advantage of the increased parallelism each new server offers. This parallelism is greater in the high-performance configuration in which a read query can be fully serviced by any one replica. For all multiple-replica configurations, Oasis achieves higher throughput than direct access to a single PostgreSQL server.

Figure 3 shows a latency breakdown for the Guide queries executed against a 2-way replicated Oasis database. The breakdown shows that read queries spend more time executing in the database than the writes. It also shows that the Oasis overhead is higher for the writes than the reads. (With two replicas, read operations can piggy-back the query on the lock request requiring fewer messages.) This figure also suggests that optimizing our XML/HTTP messaging layer could offer substantial performance gains.

## 6 Related Work

There are many existing storage management systems available to pervasive computing developers, including distributed file systems, databases, and tuple-stores. These distributed systems exhibit a variety of behaviors when clients disconnect from the network. In most systems, disconnected clients are unable to read or write data, others offer limited disconnected operation [27], while some systems give clients full read and write capabilities while disconnected [14,18,30]. We now review the storage management systems that are most relevant to Oasis and discuss how they compare.

A number of data management systems permit clients to perform updates to a local replica at any time, even when disconnected from all other replicas. These so called “update anywhere” systems are attractive because they never deny the client application the ability to write data and guarantee that the update will eventually be propagated to the other replicas. There are update-anywhere file-systems such as Coda [18] as well as update-anywhere databases such as Bayou [30] and Deno [14]. As data can be updated in multiple locations at the same time, these systems offer weaker consistency guarantees than Oasis. To achieve eventual consistency, update anywhere systems employ varying mechanisms to resolve conflicts that arise between divergent replicas. Coda [18] relies on the user to merge write conflicts that cannot be trivially resolved by the system. This technique is a poor fit for pervasive computing

environments where the user may not be near a computer to provide input or may not have the necessary level of expertise. In Bayou [30], writes are accompanied by fragments of code that travel with the write request and are consulted to resolve conflicts. While these migrating, application-specific conflict resolvers are a potentially powerful model, we believe that writing them is beyond the technical abilities of an average software engineer. Finally, Deno [14] uses rollback to resolve conflicts between replicas. Rollback is difficult to cope with in pervasive computing environment in which physical actuations take place than cannot be undone.

While peer-to-peer file sharing systems like Gnutella satisfy a number of our requirements, they do not provide a single consistent view of the data as servers connect and disconnect. Systems like Farsite [4], OceanStore [19], and CFS [8] improve on the basic P2P architecture by incorporating replication to probabilistically ensure a single consistent view. While these systems share our goals of availability and manageability, there are significant differences that make them less than ideal for pervasive computing environments. Farsite was designed for a network of PCs running desktop applications. OceanStore is geared for global-scale deployment and depends on a set of trusted servers. Finally, CFS provides read-only access to clients and is not intended as a general-purpose file system.

A few storage systems have been designed specifically for pervasive computing environments. The TinyDB system [25] allows queries to be routed and distributed within a network of impoverished sensor nodes. Systems like PalmOS allow PDA users to manually synchronize their data with a desktop computer. TSpaces [24] is one of a number of centralized tuple-based system that was written for environments with a changing set of heterogeneous devices.

Self tuning has been incorporated into several storage systems outside the domain of pervasive computing. HP AutoRaid [32] automatically manages the migration of data between two different levels of Raid arrays as access patterns change. Similarly, Hippodrome [2] employs an iterative approach to automating storage system configuration.

## 7 Future Work

The largest limitation in Oasis is the need to replicate databases in their entirety; currently an application that wants to replicate a small part of a large database needs to create an alternate database and keep it consistent. One solution is to change the level at which replication occurs to the table or possibly the record level. Another alternative would be to support partial database replicas similar to ‘views’ in SQL. We plan to investigate which alternative best fits pervasive computing as well as understand how our protocols would have to change to remain correct.

We also plan to explore how machine learning techniques can be used to guide the placement of replicas, the creation of indexes and the adjustment of the weighted-voting parameters. We believe that large gains in both availability and query performance can be attained by taking greater advantage of device characteristics and data access patterns. While work has been done in off-line self tuning, little on-line tuning research has been done for dynamic storage systems such as Oasis.

## 8 Conclusions

It is difficult to write responsive, robust pervasive computing applications using traditional data management systems. To help address this issue, we have built Oasis, a data management system tailored to the requirements of pervasive computing. Oasis presents an SQL interface and a relational data model, both of which are well suited to the data usage of typical pervasive computing applications. A peer-to-peer architecture coupled with a weighted-voting scheme provides sequentially consistent access to data while tolerating device disconnections. We have validated our initial implementation by showing it exhibits good performance as well as using Oasis to implement three typical pervasive computing applications.

## References

1. Abowd, G. D., Atkeson, C. G., Feinstein, A., Hmelo, C., Kooper, R., Long, S., Sawhney, N., and Tani, M. Teaching and learning as multimedia authoring: the classroom 2000 project. *Proceedings of ACM Multimedia '96*, 187-198, 1996.
2. Anderson, E., Hobbs, M., Keeton, K., Spence, S., Uysal, M., and Veitch, A. Hippodrome: running circles around storage administration. In *Conference on File and Storage Technology*. USENIX, 2002.
3. Arnstein, L., Sigurdsson, S. and Franza, R., Ubiquitous computing in the biology laboratory. *Journal of Laboratory Automation*, March 2001.
4. Bolosky, W., Douceur J., Ely, D. and Theimer M., Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs", In *Proceedings of ACM Sigmetrics*, 2000.
5. Brooks, R., The Intelligent Room Project. *Proceedings of the Second International Cognitive Technology Conference*, 1997.
6. Brumitt, B., Meyers, B., Krumm, J., Kern, A., and Shafer, S. EasyLiving: Technologies for intelligent environments. In *Proc. of 2nd International Symposium on Handheld and Ubiquitous Computing (2000)*, 12-29.
7. Card, S. K., Robertson, G. G., and Mackinlay, J. D.. The information visualizer: An information workspace. *Proc. ACM CHI'91 Conf.* (1991), 181-188.
8. Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, 2001.
9. Fishkin, K.P., Fox, D., Kautz, H., Patterson, D., Perkowitz, M., Philipose, M., Guide: Towards Understanding Daily Life via Auto-Identification and Statistical Analysis. *Ubihealth 2003*, Sept 2003.
10. Gifford, D. K., Weighted Voting for Replicated Data, *Proceedings of the Seventh Symposium on Operating Systems Principles*, 1979, pp. 150-162.
11. Goodman, N., Skeen, D., Chan, A., Dayal, U., Fox, S, and Ries, D., A recovery algorithm for a distributed database system, in *Proceedings 2nd ACM Symposium on Principles of Database Systems*, March, 1983.
12. Hill, J., Szewczyk, R., Woo, A., Culler, D., Hollar, S. and Pister, K.. 2000. System Architecture Directions for Networked Sensors. *ASPLOS 2000*.
13. Johanson B., Fox A. and Winograd T. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *IEEE Pervasive Computing Magazine* 1(2), April-June 2002.

14. Johanson, B. and Fox, A., The Event Heap: An Coordination Infrastructure for Interactive Workspaces, Proc. WMCSA 2002.
15. Keleher, P., Decentralized Replicated-Object Protocols. In Proc. 18th ACM Symp. on Principles of Distributed Computing, (1999), 143-151.
16. Kidd, C., Orr, R., Abowd, G.D., Atkeson, C.G., Essa, I.A., MacIntyre, B., Mynatt, E., Starner, T.E., and Newstetter, W.: The Aware Home: A Living Laboratory for Ubiquitous Computing Research. Proceedings of the Second International Workshop on Cooperative Buildings, 1999.
17. Kindberg T. and Fox A., System Software for Ubiquitous Computing. IEEE Pervasive Computing, 1(1), Jan 2002, pp. 70-81.
18. Kistler, J., Satyanarayanan, M. Disconnected Operation in the Coda File System. ACM Transactions on Computer Systems, Feb. 1992.
19. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B., OceanStore: An Architecture for Global-Scale Persistent Storage, ASPLOS, 2000.
20. LaMarca, A., Brunette, W., Koizumi D., Lease M., Sigurdsson S., Sikorski K., Fox D., Borriello G., PlantCare: An Investigation in Practical Ubiquitous Systems. UbiComp 2002: 316-332
21. Lamming, M. and Flynn, M., Forget-me-not: Intimate Computing in Support of Human Memory, in Proceedings of International Symposium on Next Generation Human Interface, (1994).
22. Lamming, M., Eldridge, M., Flynn M., Jones C., and Pendlebury, D., Satchel: providing access to any document, any time, anywhere, ACM Transactions on Computer-Human Interaction, (7)3:322-352, 2000.
23. Lamport, L. How to make a multiprocessor computer that correctly executes multiprocessor programs. IEEE Trans. on Computers, 28(9):690-691, Sept. 1979.
24. Lehman, T. J, McLaughry, S. W., Wyckoff, P., Tspaces: The next wave. Hawaii Intl. Conf. on System Sciences (HICSS-32), January 1999.
25. Madden S., Franklin M., Hellerstein J., and Hong W., The Design of an Acquisitional Query Processor for Sensor Networks. To Appear, SIGMOD, June 2003.
26. Mynatt, E., Rowan, J., Craighill, S. and Jacobs, A. Digital family portraits: Providing peace of mind for extended family members. Proc of the ACM Conference on Human Factors in Computing Systems, 2001, 333-340.
27. Oracle9i Lite Developers Guide for Windows CE, Release 5.0.1, Jan 2002.
28. Sumi, Y. and Mase, K, Digital System for Supporting Conference Participants: An Attempt to Combine Mobile, Ubiquitous and Web Computing. UbiComp 2001.
29. Rodrig M., LaMarca, A. Decentralized Weighted Voting for P2P Data Management, Third International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE 2003), Sept 2003.
30. Terry, D., Theimer, M., Petersen, K., Demers, A., Spreitzer, M. and Hauser, C. "Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System", Proc. 15th ACM Symp on Operating Systems Principles, (1995), 172-183.
31. Want, R., Pering, T., Danneels G., Kumar M., Sundar, M., Light, J.: The Personal Server: Changing the Way We Think about Ubiquitous Computing. UbiComp 2002.
32. Weiser, M., The computer for the twenty-first century. Scientific American, pages 94-100, September 1991.
33. Wilkes, J., Golding, R., Staelin, C., and Sullivan, T. The HP AutoRAID Hierarchical Storage System. ACM Transactions on Computer Systems, 14(1), Feb 1996.
34. Yang, B., and Garcia-Molina, H. Designing a super-peer network. Technical Report, Stanford University, February 2002.
35. Autonomic Computing Manifesto, [http://www.research.ibm.com/autonomic/manifesto/autonomic\\_computing.pdf](http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf), visited Mar '03.