Appears in the Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (associated with *Supercomputing* 2005), November 8, 2004. Pittsburgh, PA.

Mobile OGSI.NET: Grid Computing on Mobile Devices*

David C. Chu
EECS Department
University of California, Berkeley
Berkeley, CA 94720-1776
davidchu@eecs.berkeley.edu

Marty Humphrey
Department of Computer Science
University of Virginia
Charlottesville, VA 22904
humphrey@cs.virginia.edu

Abstract

The problem with the Grid is that it does not currently extend completely to devices, because these devices are not viewed as having sufficient capability to be both clients and services. We design, implement and evaluate Mobile OGSI.NET, which extends an implementation of grid computing, OGSI.NET, to mobile devices. Mobile OGSI.NET addresses the mobile devices' resource limitations and intermittent network connectivity, factors which differentiate them from traditional computers. Because Mobile OGSI.NET uniquely supports the hosting of Grid Services on the device, Mobile OGSI.NET is an important step toward making the mobile device a first-class entity in Grids based on OGSI or the Web Services Resource Framework (WSRF).

1. Introduction

Mobile electronic devices such as Personal Digital Assistants (PDAs), Smart Phones, and wearable computers, are increasingly common. Individuals will frequently own a collection of these mobile devices. Yet, these devices are often resource limited: processing power is low, battery life is finite, and storage space is constrained. These restrictions slow application execution, and hinder operability.

Arguably, applications executing on devices must be made aware of concurrently-executing applications in order to optimally use the limited resources. Previous related work suggests several approaches to address this problem. We categorize these approaches as mobile collaborative computing tools; single-device resource management; and multi-device grid computing resource management.

Mobile collaborative computing tools ease development of collaborative applications. For example, iMobile [1] defines enterprise services for secure mobile device access. The YCab toolkit [3] allows ad-hoc mobile device collaboration. Quickstep [2] provides synchronous collaboration abilities for mobile devices. Yet, mobile collaborative computing often restricts the mobile device to a portal. The mobile device then depends upon external computers, such as traditional desktops and servers.

Several systems attempt to scale the mobile experience in response to the fluctuation of resources on a single device. Odyssey [4] implements a viceroy gateway as an intermediary between the critical resources and wardens acting on behalf of user applications. In particular, Odyssey varies application fidelity in response to changes in network bandwidth availability. Flinn and Satyanarayanan also extended Odyssey to respond to changes in battery power [5]. Similarly, the DQM [6] varies the quality of a toy application in response to processor availability. These single device approaches do not leverage the additional resources of other available computers. As a result, these approaches must degrade the user experience.

Grid computing offers an attractive alternative for resource-demanding applications. *The paradigm of Grid Computing as applied to resource limited devices is that somehow the devices can collectively deliver the quality of service needed by the end-user*. Currently, grid computing predominately serves computationally intensive scientific and enterprise applications and operates on cluster computers or supercomputers [7]. The widely-used Globus Toolkit version 2.x (GT2) [8] provides mechanisms for constructing a grid infrastructure. Legion [9] offers a similar platform for grid computing. However, both of these systems used

* This work was supported in part by the National Science Foundation under grants ACI-0203960 (Next Generation Software program), the National Partnership for Advanced Computational Infrastructure (NPACI), and Microsoft Research.

proprietary communication interfaces. Proprietary interfaces limit interoperability and extensibility, especially to new platforms such as personal mobile devices.

The architects of Globus, wishing to define grid computing in term of web services [10][11], developed the Open Grid Services Architecture (OGSA) [12]. The Open Grid Services Infrastructure (OGSI) [13], a normative specification, quickly followed. Collectively these define grid services, extensions to the SOAP communications protocol for grid computing. This provides true platform-independent grid computing.

Currently, OGSI implementations exist for several platforms, or runtimes. Sandholm et al. implement OGSI for the Java Virtual Machine runtime [14]. Humphrey et al. implement OGSI for the Microsoft .NET Framework runtime [15]. However, very few mobile devices can support either of these runtimes. Rather, many mobile devices run Windows CE with the .NET Compact Framework, a substantially stripped-down version of the .NET Framework. In addition, neither of these implementations considers the addition of mobile device constraints, such as limited resources and intermittent network connectivity.

Several efforts combine grid computing and mobile devices. Gonzalez-Castano incorporates mobile devices into Condor as client front-ends for job submission and job querying to traditional supercomputer grids [16]. Phan et al. suggest a proxy-based cluster architecture for introducing mobile devices into traditional grids [17], though provides no implementation for evaluation. Clarke and Humphrey investigate the challenges of integrating mobile devices into the Legion grid computing system [18]. While addressing some of the particular concerns of mobile devices, none of these efforts embraces the community-adopted OGSI specification.

In this paper, we describe Mobile OGSI.NET, created to promote resource sharing and collaboration that improves the user experience. Mobile OGSI.NET extends an implementation of grid computing, OGSI.NET [15], to mobile devices. By adhering to the OGSI specification, Mobile OGSI.NET interoperates with existing OGSI implementations, such as GTK and OGSI.NET. Mobile OGSI.NET also addresses the mobile devices' resource limitations and intermittent network connectivity, factors which differentiate them from traditional computers. Because the Web Services Resource Framework (WSRF) is a re-factoring of OGSI, we believe that many of the results achieved during the implementation of Mobile OGSI.NET will be important to WSRF-based Grids as well.

The outline of this paper is as follows. Section 2 gives the goals and requirements of Mobile OGSI.NET. Section 3 presents the software architecture. Section 4 contains a description of the implementation and optimization. Section 5 contains an evaluation of Mobile OGSI.NET in a particular context that we believe is representative of future device usage in Grids. Section 6 is the conclusion.

2. Mobile OGSI.NET: Goals and Requirements

Before describing the architecture of Mobile OGSI.NET, we outline the four design goals and provide justification for the goals. First and foremost, we wished to construct a platform that provides the better potential for collaboration among mobile devices. That is, we wish to facilitate a collection of applications---on a single device or on multiple devices---being able to work together on a particular problem. This is one version of the Grid problem. We believe that a common set of protocols and software can facilitate this, particularly as a substrate by which

Second, we aim to support this style of collaboration among mobile devices with traditional, non-mobile workstation computers and server computers. While peripheral to the main goal of mobile to mobile collaboration, mobile to desktop/server collaboration opens further possibilities of maximizing resource utilization and improving user experience. Mobile devices offer convenience and contextual relevance while desktops and servers offer comparatively limitless resources and networking.

Third, the collaboration architecture should operate on many device platforms. Mobile devices present widely varying hardware interfaces. To enable practical mobile device collaboration, we must implement Mobile OGSI.NET upon a widely used operating system. If we select an operating system without critical-mass deployment, most mobile devices would be incapable of running Mobile OGSI.NET.

Fourth, the collaboration architecture must address the particular characteristics of mobile devices. Mobile devices experience intermittent networking and resource constraints. Due to mobility, networking quality and availability fluctuate as the user travels with the device. Naïve schemes which rely on the availability of particular network functions will fail under such conditions. Resource constraints constantly hinder the user experience. Familiar resources include processing power, battery life and storage capacity. However, we broaden traditional notions of resources to include any hardware capability needed to perform

the user's desired task. This may include display screens for visual applications, or display screens and speakers for multimedia applications. For example, if the user may wish to check stock prices while simultaneously viewing a streaming video. In this scenario, even if the user possesses multiple display screens (for example a PDA screen and wristwatch screen), she is limited to sequential viewing. Mobile OGSI.NET should optimize resource usage on behalf of the user.

After identifying the design goals, we translated these goals into requirements. First, collaboration inherently involves agreement on a set of protocols and behaviors. The Open Grid Services Infrastructure (OGSI) Specification [13] for grid computing emerged as the preferred mechanism. While traditional servers have few similarities with mobile devices, implementing the server-based OGSI specification on resource-limited devices offers several advantages. First, we were already familiar with OGSI through our project's implementation on .NET, OGSI.NET. Second, conformance with the OGSI Specification allows Mobile OGSI.NET to interoperate with desktops and servers running the OGSI-based Globus Toolkit or OGSI.NET[2]. Neither the Globus Toolkit nor OGSI.NET runs on mobile devices. The alternative to the OGSI Specification was adopting a custom communication protocol. This choice would not allow immediate interoperability with desktops and servers, without further custom desktop and server software.

Second, we chose to implement Mobile OGSI.NET on the Microsoft PocketPC 2003 operating system. PocketPC 2003 is the latest edition of the PocketPC family operating systems. The other dominant mobile device operating systems are Palm OS and Linux. By some projections, the Pocket PC operating systems will compromise 40% of market in 2007. Note that this includes devices ranging from embedded devices to cell phones to ultra-small laptops.

In addition, we chose to implement Mobile OGSI.NET on top of the .NET Compact Framework [19]. This runtime acts as an intermediary software layer between the application and the operating system. This layer primarily offers a convenient GUI. Mobile OGSI.NET itself does not need a GUI. However, we chose the .NET Compact Framework because we wished to take advantage of several useful utility libraries and applications developed on the .NET Compact Framework. Figure 1 illustrates the relationship between the various hardware and software layers described thus far.

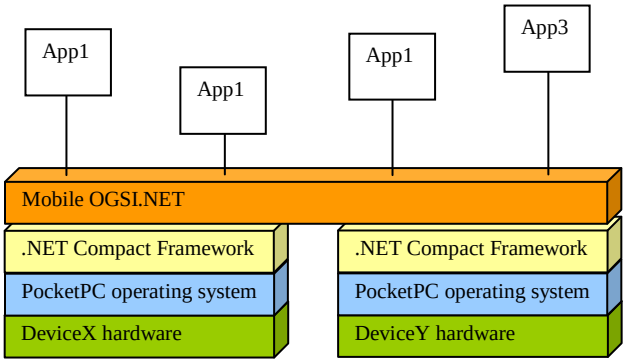

Figure 1: Mobile OGSI.NET and its relation to other device hardware/software layers. Mobile OGSI.NET bridges multiple devices.

Third, specifically to accommodate computing platforms that are quickly shut off, we require process migration and distributed execution capabilities. Mobile OGSI.NET should migrate processes away from resource depleted devices. Distributed execution allows the user's job to run among several devices, with each device handling some part of the job. These provide the building blocks for improving resource utilization. Our specific short-term goal is to have the resource owner engage this process migration mechanism directly; our longer-term goal is to silently migrate processes as we notice, for example, that battery reserve is becoming dangerously low.

3. Mobile OGSI.NET Architecture

The Mobile OGSI.NET architecture consists of three main layers: Monash University Mobile Web Server [20], the Grid Services Module, and the Grid Services. Each layer handles a separate concern. The Mobile Web Server handles endpoint to endpoint message reception and transmission. The Grid Services Module handles Grid Services message parsing and multiplexes messages to the appropriate Grid Service. The Grid Service handles application logic and processing. Figure 2 illustrates this system architecture.

3.1 Mobile Web Server

The Mobile Web Server, an HTTP server developed at Monash University, handles endpoint message composition for sends and reconstruction for

[2] In certain instances, GT and OGSI.NET use message parameters unspecified in OGSI Specification. We developed Mobile OGSI.NET to conform to these de facto specification elements as well.

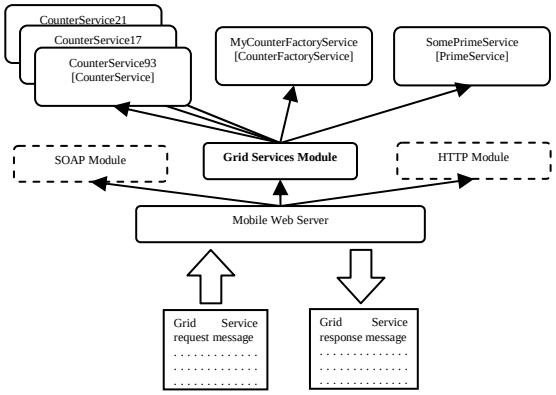

Figure 2. Mobile OGSI.NET Architecture

receives. The Mobile Web Server functions both for traditional HTTP requests for web content and advanced SOAP requests for Web Services. The Mobile Web Server acts as a demultiplexer of incoming messages for the appropriate processing module, such as the HTTP Module and SOAP Module. This makes the Mobile Web Server particularly attractive as a foundation from which to develop Grid Services; minimally, we add a new Grid Services Module.

The alternative HTTP servers to the Mobile Web Server are the Microsoft PocketPC 2003 Web Server and several commercial web servers. We chose the Mobile Web Server because the Mobile Web Server is open source, already contains an example SOAP module, and supports module add-ons. None of the alternatives provided these features.

3.2 Grid Services Module

We developed the Grid Services Module to handle the core processing necessary for Grid Services. The Grid Services Module parses the HTTP request content as a SOAP message, and then redirects the message to the appropriate Grid Service. Just as the Mobile Web Server demultiplexes messages to the appropriate module, the Grid Services Module demultiplexes messages to the appropriate Grid Service. The service request address keys the demultiplexing procedure.

To achieve the Grid Service transient and stateful properties, we support dynamic service instantiation and service querying, in addition to standard to standard web service operations. The OGSI Specification provides a detailed factory interface description. We implement factories simply as Grid Services which subscribe to this interface. For example, a CounterFactoryService provides the createService port which creates CounterService instances. We allow factories to share a central registry in order to maintain a consistent view of a server's services. A configuration file, gridservice.web, specifies the initial Grid Services available upon host initialization.

We currently do not implement a method to perform dynamic service discovery. Traditional web service discovery techniques, such as WSDL retrieval, still apply. However, clearly users can not discover dynamic services with this technique without a priori knowledge of currently instantiated services. We plan to address this need by implementing a host services query feature.

3.3 Grid Services

Grid Services contain application logic and are simply .NET Compact Framework Dynamic Link Libraries (DLLs). Application developers may independently build application specific DLLs for use as Grid Services. Following the SOAP module, we allow the application developer to specify which methods are web service methods with the [WebService] attribute. For example, the CounterService code listing in Figure 2 illustrates how regular methods are decorated with attributes. This method mimics traditional web services development; the experienced web services application developer should find this programming model comfortable. Figure 2 lists the Grid Services implemented in Mobile OGSI.NET that are discussed in the rest of this paper.

```
public class CounterService : BaseService {
  int counter;
  public CounterService() {
        counter = 0;
  }

  [WebMethod]
  public int add(int value) {
        counter += value;
        return counter;
        // This method is decorated with the
        // [WebMethod] attribute and is accessible
        // from Mobile OGSI.NET
  }
  public void nonWebServiceMethod() {
        // This method lacks attribute decorations
        // and hence is not accessible from
        // Mobile OGSI.NET
  }
}
```

Figure 2: Grid Services web method declaration using attributes

Table 1: Grid Services currently implemented. These demonstrate the breadth of possible services.

Grid Service Type	Description	Web Methods
Hello World Service	Archetypal, stateless web service; does not actually use any Mobile OGSI.NET features	SayHelloTo()
Counter Service	Archetypal Grid Service; keeps basic state, an integer counter.	addValue()
Counter Factory Service	Creates Counter Service services.	createService()
Mobile Counter Service	Mobile Grid Service; can migrate between hosts.	addValue()
Prime Service	Distributed Grid Service; takes advantage of other available devices.	Search()
Prime Worker Service	Works on behalf of a Prime Service	IsPrime()
Prime Worker Factory Service	Creates Prime Worker Service services on behalf of Prime Service.	createService()
Group Service	Manages group dynamics among a collection of devices.	addMember() removeMember() getMembers()

4. Mobile OGSI.NET Implementation

4.1 Mobile Web Server

We made several optimizations and improvements to the Mobile Web Server. First, we extended the Mobile Web Server's demultiplexing capabilities. The Mobile Web Server originally only allowed keying on file extension type. This limited capability restricted the naming of dynamic Grid Services. We added service path-based routing logic so that any request address with the /OGSA/Services prefix is processed as a Grid Service request. The OGSI.NET hosting environment's ISAPI filter performs a similar function. Second, the Mobile Web Server often failed to correctly reconstruct HTTP messages consisting of multiple segmented TCP packets. We fixed this shortcoming. Lastly, the Mobile Web Server's current architecture supports only one request processing at a time whereas traditional web servers may support thousands of simultaneous requests. This blocking model forces us to treat localhost requests as a special case, and prevents multiple request handling. While at this time we have not modified the Mobile Web Server to support simultaneous requests, we look forward to modifying the Mobile Web Server to a thread pool model.

4.2 Grid Services Module

Our first Grid Services Module, based on the SOAP Module, lacked several important features. First, the original Grid Services Module lacked support for SOAP headers. We implemented support for SOAP headers. This allows the Grid Services Module to support the GXA family of protocols such as WS-Security and WS-Addressing [24]. Currently the Grid Services Module uses the WS-Addressing "To" address as the demultiplexing key for the appropriate Grid Service. This allows routable service requests and responses. Similarly, we envision support for WS-Security message authentication and encryption.

Second, the original Grid Services Module did not support complex parameter and return types for web service methods. For example, a `MyContactsService` service may have an `AddContact()` method. The application developer prefers to pass a single Contact object rather than a semantically unorganized list of ContactName, ContactPhone ContactAddressStreet, ContactAddressCity, etc. We extended parameter types to permit complex types. We leave the less frequently used complex return types for future work.

4.3 Mobile Grid Services

Battery power severely constrains individual devices. Therefore, individual devices tend to lack reliability. Battery power exhaustion or intentional power management (automatic/manual device power off) increases the failure likelihood. To address this problem, Mobile OGSI.NET allows services to migrate from host to host. Specifically, we implemented basic Grid Service state saving and loading. A Grid Service may save its state which another Grid Service of the same type can then load and continue running. For example, MyCounterService291 of type CounterService may save its state via SaveCounterState(). Then, another MyCounterService18 also of type CounterService loads the state object via LoadCounterState() and continues running as if resuming from the previous save point.

The basic migration procedure demands several improvements. First, we can provide an OGSI base service and interface. These might export SaveInstance() and LoadInstance() methods with Grid Service extensibility elements. This allows service

agnostic migration. For example, if a host detects it is about to fail, it may migrate all of its services to other hosts by using this generic interface.

Second, currently the user initiates both save and load calls. We plan to remove user involvement from the migration process via a GroupManagerService, which either periodically or on an event basis, examines host and service metadata. This examination may reveal that certain Grid Services should migrate. For example, upon detecting less than 20% battery power remaining on HostA, the GroupManagerService initiates calls to migrate ServiceX on HostA to HostB. Subsequently clients, after finding ServiceX absent from HostA, query the GroupManagerService and discover ServiceX moved to HostB. In OGSI specification terms, the GroupManagerService is the handle resolver and the address to HostA or HostB is the Grid Service Reference.

Third, we can compare Grid Services migration to process checkpointing. In the example above, the checkpoint consists solely of the current counter value. This simple scalar value is trivial to encode and transmit. However, a Grid Service may be involved with open files, socket connections and other localhost resources. This complex, host and platform-specific state has been notoriously difficult to capture and use in traditional checkpointing. If the need arises, we plan to employ checkpointing and migration processes used in grid computing frameworks [21].

4.4 Distributed Grid Services

Traditionally, multiple applications fight for scarce resources on a single device. Mobile OGSI.NET harnesses the network of devices to permit distributed application execution. We allow a Grid Service to distribute its work among multiple hosts via the Grid Services protocol. PrimeService is our prototype distributed Grid Service example. PrimeService.Search() finds all primes in a user specified range. This problem decomposes into primality testing each integer in the specified range. Our goal is to distribute this task among multiple devices.

We follow a *manager/worker* model for distributing this service. First, when a client request arrives, `PrimeService`, the manager, obtains a listing of the other available devices. In the current implementation, these peers are known a priori. Second, having established peer devices, the `PrimeService` creates `PrimeWorkerService` services on the peers via calls to `PrimeWorkerFactoryService` on the peers. These newly created workers are solely for the use of the manager. Third, the `PrimeService` gives a portion of work i.e. primality testing a particular integer range, to each `PrimeWorkerService`. As workers report results, the manager gives each worker another piece of the job until all integers have been handed out. Finally, the manager reports the summary result to the client when all integers in the range have been tested.

The manager/worker model both decreases job completion time and more fairly distributes resource consumption, which are keys in a resource-limited environment. However, distributed applications introduce several challenges for the application developer. First, managers must distribute job pieces to multiple workers asynchronously. This contrasts from traditional, comfortable synchronous programming. Second, distributed programs must gracefully handle complex failure modes. Failures may occur both because of network connectivity or process failure. These factors considerably increase programming complexity. We do not investigate these problems in depth in Mobile OGSI.NET. Currently, application developers must implement all asynchronous logic and handle all failure modes.

Workers must have appropriate services available i.e. `PrimeWorkerFactoryService` in order to assist `PrimeService`. More generally, service logic should be transported and deployed to the devices that require the service. The current Mobile OGSI.NET implementation assumes `PrimeWorkerFactoryService` availability at any reported peer; all Grid Services we expect to use (generally factories) are deployed along with Mobile OGSI.NET. If we relax this assumption, we then require mobile code. We do not investigate mobile code in depth in Mobile OGSI.NET.

4.5 Group Management

The mobile service and distributed service features we have discussed involve group creation and management. Mobile OGSI.NET provides GroupService, a Grid Service for creating and managing groups. GroupService differs from other services in that it is not dynamic. A client can rely on the existence of GroupService at the same path, /OGSA/Services/GroupService, for any device running Mobile OGSI.NET. GroupService allows new services to join, old services to leave, and queries about who is in the group. This suffices to provide possible migration targets for mobile services, and a list of candidate workers for manager services. Additionally,

we design the GroupService interface to allow Bluetooth [23] to initiate group membership actions.

The current basic implementation of GroupService does not optimally perform group creation and management, nor does it handle the many failure scenarios possible in groups. For example, membership change may not be detected simultaneously by all members. We consider this in future research.

5. Evaluation

We tested Mobile OGSI.NET on a collection of PocketPCs consisting of one HP iPAQ and two Compaq iPAQs. The HP iPAQ ran the PocketPC 2003 operating system with 400MHz Intel XScale processor, 128 MB RAM and 48 MB ROM. The two Compaq iPAQ 3670s ran the PocketPC 2000 operating system with 206MHz Strong Arm processor, 64 MB RAM and 16 MB ROM[3]. The HP outperformed the Compaq by a factor anywhere from 1.7 to 3.5, depending upon the operation. The Mobile OGSI.NET server (Mobile Web Server and Grid Services Module) amounted to 147 KB. The sample Grid Services described in Table 1 occupied an additional 40 KB. In total, the deployment occupied 187 KB. This insidiously small footprint assumes the .NET Compact Framework, at 4400 KB, already resides on the device. This holds true for most newer PocketPCs, such as the HP, but does not hold for older PocketPCs, such as the Compaqs.

We conducted the large majority of tests by averaging over three samples. Variance was generally small enough to justify only three samples. This does not include the first test run which we always discard. The first run often executes twice to three times as slowly as subsequent runs, likely due to the just-in-time compilation of .NET Framework applications. We disabled automatic user non-interaction standby and screen dimming.

5.1 Standard Services Performance

We compare the performance of Mobile OGSI.NET on PocketPCs to OGSI.NET on a traditional desktop. Mobile OGSI.NET implements a far smaller subset of the OGSI Specification than OGSI.NET. We choose to test the basic CounterService and CounterFactoryService services. Figure 3 shows the Mobile OGSI.NET performance on the three different platforms, the HP iPAQ, the Compaq iPAQ and a desktop machine. All clients were running on a wireless PocketPC.

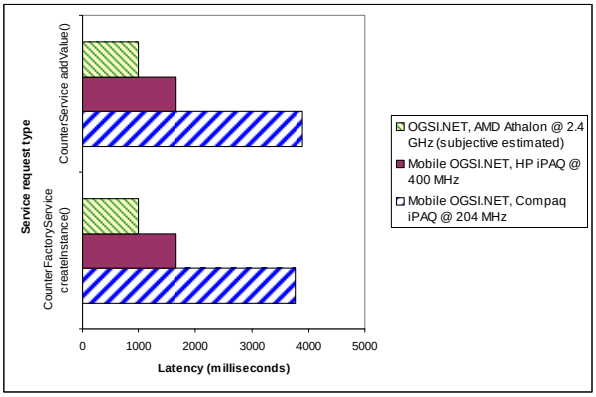

Figure 3: Latency for basic services, CounterService and CounterFactoryService.

Mobile OGSI.NET latency compares favorably to OGSI.NET latency. This may appear surprising at first, given the large processing power disparity. However, for these basic operations, network latency dominates actual processing time. Additionally, all response times (sub 4 seconds) allow reasonably interactive user experiences.

5.2 Distributed Services Performance

We measure the performance gain in response to increased hardware resources. In these tests, we investigate two scenarios types. In the first scenario type, the single HP iPAQ runs a traditional, non-distributed prime searching application. This case allows us to benchmark any performance gains or losses.

In the second scenario type, the single PocketPC emulator searches for primes by distributing work to a collection of iPAQs. We always use at least the one HP iPAQ and zero, one or two Compaq iPAQs. Note that the emulator does no actual work besides initiating service requests and collecting results.

The prime search occurs starting from 100,000 includes as many as the next 400,000 integers. This relatively tame search space provides enough distinction to evaluate performance behavior. Figure 4 graphs the behavior of the four scenarios.

[3] The XScale and Strong Arm processor speeds are comparable since the XScale adopts the Arm instruction set.

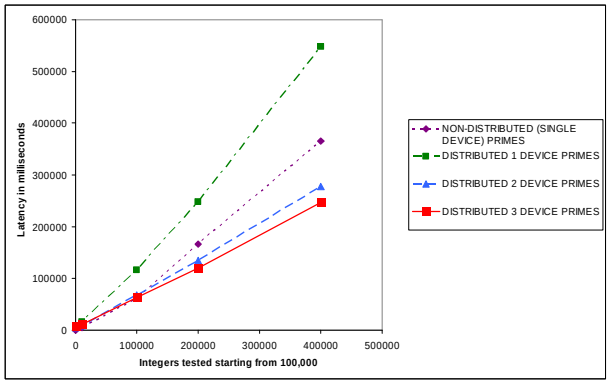

Figure 4: Latency with various numbers of hardware resources.

Response time does indeed improve with multiple devices. Yet we also observe several interesting phenomena. First, the distributed prime search does not scale well with increasing search space. For example, at a search space of 400,000, the distributed one device prime search performs 50% slower than the non-distributed search. Ideally, we would observe only a small constant overhead for `PrimeWorkerService` creation. Similarly, the distributed three device prime search performs only 48% faster than the non-distributed version. Ideally, we would observe a three fold[4] increase in performance. However, the prime search service must transmit every prime found with increasingly larger search space requests. This enormous quantity of data (logarithmic in the size of the search space) segments packets and causes increased network latency.

Second, for small jobs (under 100,000 search space), the local non-distributed search far outperforms any distributed service. Also, for very small jobs (under 10,000), fewer devices perform better than more devices. The PrimeWorkerService creation overhead explain these results.

Lastly, three distributed devices only slightly outperform two distributed devices in the largest search space (12% faster). We expect this performance disparity to widen as the search space grows.

Next, we investigate the average battery usage per device in the same four scenarios. Figure 5 shows the resulting energy drain in the HP iPAQ. The Compaq iPAQs had battery meter granularities too imprecise for this comparison. Greater job distribution does indeed more evenly distribute battery usage than less job distribution. The non distributed search appears to jump around the distributed one device search; we attribute this to battery meter calibration imprecision. Also, while more distributed devices may use less power per device, the total power usage for the entire job is greater. This as expected since, as mentioned previously, the distributed searches spend a good deal of time just reporting back results.

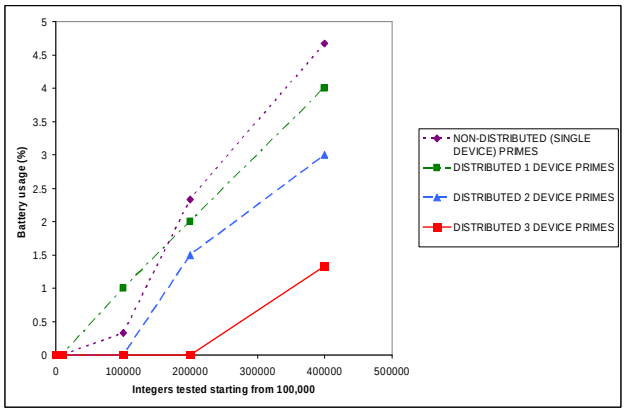

Figure 5: Battery usage with various numbers of hardware resources

6. Conclusion

We have designed and implemented Mobile OGSI.NET, an OGSI Specification-conformant grid computing hosting environment. To the best of our knowledge, we are the first to offer OGSI Grid Service hosting on small devices.

Furthermore, we have developed specific solutions for the mobile, resource-constrained environment. Our implementation occupies minimal footprint yet supports arbitrary application-specific Grid Services. Mobile Services may migrate during deployment in response to local or global events. Distributed Services better utilize available resources and prolong the lifetime of individual devices.

Mobile OGSI.NET makes initial advances towards multiple device collaboration. At the same time, we have bridged two very disparate fields: we have taken high performance supercomputing designs and adapted these for personal mobile devices. This fruitful investigation has yielded a hosting environment that can interoperate with the spectrum of computing resources.

However, we see several ways to improve Mobile OGSI.NET. First, Mobile OGSI.NET does not currently implement Service Attributes, Grid Notifications, nor security mechanisms.

Second, we need to loosen the restrictions of the GroupManagerService; it cannot currently handle the truly dynamic environment we anticipate mobile

[4] Though not quite three fold since the Compaq iPAQs are not nearly as powerful as the HP iPAQ

devices operating. This is a non-trivial investigation that will require a significant study.

Third, Mobile OGSI.NET should port easily to other mobile and non-mobile embedded devices in the Windows CE operating system family. This will allow Mobile OGSI.NET to coordinate not only PocketPCs, but varied other embedded devices as well. In pursuit of this goal, we have built a minimal .NET Compact Framework Windows CE platform.

Fourth, Bluetooth networking [23] integration may provide Mobile OGSI.NET with very desirable ad hoc capabilities. Bluetooth's ad hoc properties allow a user's set of mobile devices to collaborate with minimal configuration. In addition, mobile devices may use Bluetooth networking regardless of IP networking loss. Mobile devices frequently experience IP networking loss due to mobility and the non-universal coverage of IP access points. Bluetooth integration may be particularly challenging because the OGSI Specification is built upon IP networking.

Fifth, we have yet to investigate resource sharing among different users. We may approach this from either a game theoretic formulation or policy perspective. Both the grid computing and web services community are working towards developing nascent policy-based approaches.

Lastly, we will look to apply our experiences with Mobile OGSI.NET in designing and implementing Mobile WSRF.NET. As devices become increasingly capable, we believe that they will both be consumers and producers of WSRF-compliant grids.

7. References

[1] D. Buszko, W. Lee, and A. Helal. "Decentralized Ad-Hoc Groupware API and Framework for Mobile Collaboration." *Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work*. 2001.

[2] J. Roth and C. Unger. "Using Handheld Devices in Synchronous Collaborative Scenarios." *Personal and Ubiquitous Computing*. Volume 5, Issue 4, December 2001.

[3] Y. Chen, H. Huang, R. Jana, T. Jim, M. Hiltunen, S. John, S. Jora, R. Muthumanickam and B. Wei. "iMobile EE – An Enterprise Mobile Service Platform." *Wireless Networks*. Volume 9, Issue 4, July 2003.

[4] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker. "Agile application-aware adaptation for mobility." *ACM SIGOPS Operating Systems Review , Proceedings of the sixteenth ACM symposium on Operating systems principles*. Volume 31, Issue 5, Octoboer 1997.

[5] J. Flinn, and M. Satyanarayanan. "Energy-aware adaption for mobile devices." *ACM Symposium on Operating Systems Principles. Proceedings of the seventeenth ACM symposium on Operating systems principles*. 1999.

[6] S. Brandt, G. Nutt, T. Berk. and M. Humphrey. "Soft real-time application execution with dynamic quality of service assurance." *Quality of Service, 1998. (IWQoS 98) 1998 Sixth International Workshop*. 18-20 May 1998.

[7] M. Baker, R. Buyya, and D. Laforenza. "Grids and Grid technologies for wide-area distributed computing" *Software - Practice and Experience*. 2002.

[8] I. Foster and C. Kesselman. "Globus: A Metacomputing Infrastructure Toolkit." *International Journal of Supercomputer Applications*. Vol. 11, Issue 4, 1997.

[9] A.S. Grimshaw, A.J. Ferrari, F.C. Knabe and M.A. Humphrey, "Wide-Area Computing: Resource Sharing on a Large Scale," IEEE Computer, 32(5): 29-37, May 1999.

[10] I. Foster, C. Kesselman, and S. Tuecke. "The Anatomy of the Grid - Enabling Scalable Virtual Organizations." *International Supercomputer Applications*. 2001.

[11] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. "The Physiology of the Grid." *Global Grid Forum*. June 2002.

[12] I. Foster, and D. Gannon. "The Open Grid Services Architecture Platform." *Global Grid Forum Drafts*. http://www.ggf.org/ogsa-wg. February 2003.

[13] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm., D. Snelling, and P. Vanderbilt. "Open Grid Services Infrastructue (OGSI) Version 1.0." *Global Grid Forum Drafts.* http://www.ggf.org/ogsi-wg. April 2003.

[14] T. Sandholm, S. Tuecke, J Gawor, R. Seed, T. Maguire, J. Rofrano, S. Sylvester, and M. Williams. "Java OGSI Hosting Environment Design - A Portable Grid Service Container Framework." *Global Grid Forum Drafts* http://www.gridforum.org/Meetings/ggf7/drafts/OGSI%20Java%20Hosting%20Environment12.pdf. March 2003.

[15] G. Wasson, N. Beekwilder, M. Morgan, and M. Humphrey. OGSI.NET: OGSI-compliance on the .NET Framework. In *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid.* April 19-22, 2004. Chicago, Illinois.

[16] F. Gonzalez-Castano, J. Vales-Alonso, and M. Livny. "Condor Grid Computing from Mobile Handheld Devices." *Mobile Computing and Communications Review*. Vol. 6, No. 2. *ACM SIGMOBILE Mobile Computing and Communications Review*. Volume 6 , Issue 2, April 2002.

[17] T. Phan, L. Huang, and C. Dulan. "Integrating Mobile Wireless Devices Into the Computational Grid." *Mobicom 2002*. 2002.

[18] B. Clarke, M. Humphrey. "Beyond the 'Device as Portal': Meeting the Requirements of Wireless and Mobile Devices in the Legion Grid Computing System." *2nd International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and*

Mobile Computing (associated with IPDPS 2002), Ft. Lauderdale, April 19, 2002.

[19] .*NET Compact Framework.* http://msdn.microsoft.com/vstudio/device/compactfx.aspx. Microsoft. 2002.

[20] N. Nicoloudis and D. Pratistha. .*NET Compact Framework Mobile Web Server Architecture*. Microsoft MSDN Library. July 2003. http://msdn.microsoft.com/library/en-us/dnnetcomp/html/NETCFMA.asp.

[21] *Condor Checkpointing*. Condor High Throughput Computing Project. Visited March 2004. http://www.cs.wisc.edu/condor/checkpointing.html.

[22] T. Thorn. *Programming Languages for Mobile Code*. ACM Computing Surveys, Vol. 29, No. 3. September 1997.

[23] *Bluetooth*. Visited March 2004. http://www.bluetooth.com.

[24] D. Ferguson, T. Storey, B. Lovery and J. Shewchuk. *Secure, Reliable, Transacted Web Services: Architecture and Composition*. IBM/Microsoft Whitepapers. September 2003.