An Architecture for e-Science and its Implications

Geoffrey Fox[1,2,4], Hasan Bulut[1,2], Kangseok Kim[1,2], Sung-Hoon Ko[1], Sangmi Lee[5], Sangyoon Oh[1,2], Xi Rao[1,2], Shrideep Pallickara[1], Quinlin Pei[1,2], Marlon Pierce[1], Ahmet Uyar[1,4], Wenjun Wu[1], Choonhan Youn[1,3]
Dennis Gannon[2], and Aleksander Slominski[2]

[1]Community Grid Computing Laboratory, Indiana University
501 N Morton Suite 224, Bloomington IN 47404
gcf@indiana.edu, hbulut@indiana.edu, kakim@cs.indiana.edu, shko@grids.ucs.indiana.edu,
ohsangy@cs.indiana.edu, xirao@indiana.edu, spallick@indiana.edu, qpei@indiana.edu,
marpierc@indiana.edu, wewu@indiana.edu,
[2]Computer Science Department, Indiana University
gannon@cs.indiana.edu, aslom@cs.indiana.edu
[3]EECS Department, Syracuse University
auyar@syr.edu, cyoun@indiana.edu
[4]School of Informatics and Physics Department, Indiana University
[5]Computer Science Department Florida State University
slee@grids.ucs.indiana.edu

Keywords: Web Service, Computational Science, e-Science, Event Service, JXTA, JMS, Java Message Service, Peer-to-Peer

Abstract

We describe Peer-to-Peer Grids built around Integration of technologies from the peer-to-peer and Grid fields. We focus on the role of a powerful event services using uniform XML interfaces and application level routing. We describe a research system Narada and compare with JXTA (Peer-to-Peer) and JMS (Java Message Service) giving some initial performance results.

1 e-Science

Over the last decade or so, there has been dramatic progress in the application of computing to scientific and engineering research. There were several major initiatives such as the HPCC (High Performance Computing and Communications) program and the set of NSF Grand Challenge projects, which largely focused on this. The parallel computing emphasis of the early nineties has shifted with newer programs such as ITR (Information Technology Research) from NSF. The new centerpiece is research in

Fig. 1: A Paradigm for Computational Science and Information Technology

areas like grids [1,2] and distributed computing, which overshadows that on classic computational science. Now NSF has a new cyberinfrastructure report [3] continuing the same emphasis on distributed systems and their broad impact on scientific research. Already the United Kingdom has a major effort [4] in this area for which the term e-Science has been coined.

How can modern computing (information technology) change and support Science. As one example, take the National Virtual Observatory [5], which eloquently illustrates the new approach in a way that can be generalized from astronomy to other fields. Just a few years ago, astronomy largely involved individual groups designing new instruments, developing particular observing strategies and managing the data taking on some instrument either in the space or on the ground – this data was lovingly analyzed to discover and publish new scientific insights. This "stove-pipe" approach to observational (experimental) science is typical in many fields. It makes certain that those who build the equipment can ensure their data is properly interpreted with the usually difficult instrumental corrections properly applied. Note that it is usual to compare "reasonably corrected experimental data" with hypotheses (models) to which other instrumental effects are applied. Often it is not possible to remove all instrumental effects from a dataset so that it can be compared with a pristine theory. Nevertheless we imagine a new astronomy where our investigator has the best analysis and visualization capabilities connected to the global internet. This analysis will integrate the data from multiple instruments spanning multiple wavelengths and multiple regions of the sky. This vision stresses the high performance networks, data access, management and processing enabled by "information technology". A similar story can be told in other fields. Bio-informatics involves the integration of gene data banks scattered around the Internet; visionaries imagine this extended to include massive data associated with individuals and enabling a new personalized medicine. High energy and nuclear physics experiments will involve thousands of physicists analyzing petabytes of data each year coming from a new generation
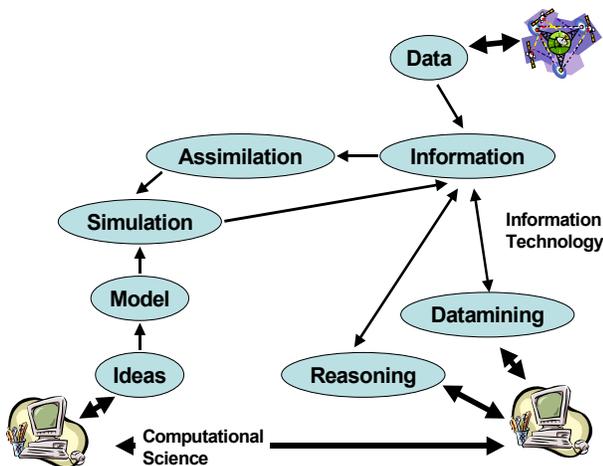
of detectors at high intensity accelerators such as the LHC at CERN. Fields such as climate, environment, earthquake and weather will also benefit from what has been termed the data deluge. Note the gathering of data and its computer-based analysis is not new – in fact the World Wide Web originated in CERN from a tool to support transcontinental high energy physics collaborations. Rather the scale (amount of data) and breadth (integration of datasets) has changed dramatically.

We can simplistically summarize the situation this way. Computational Science was built on the vision that computers would represent a virtual laboratory where one could explore new concepts from simulations and comparison of these with experimental data. The very successful agency supercomputers (NSF, DoD, DoE, NASA, NIH and others) have supported a growing interest in this mode of computational science. We understood that Moore's law would inevitably drive this field with steadily increasing simulation, storage and networking performance. Now we see that advances in device physics are driving both computer and instrument performance. Thus the data deluge is re-invigorating and re-shaping observational fields. In fact as shown in fig. 1 one can and should integrate these themes. Sensors will produce a lot of information but so also will simulations. Various techniques: visualization, statistical analyses, "datamining" will extract knowledge from the information gleaned from simulations and raw data sources These will be fed back into theoretical science and so the classic collaboration between the twin pillars – theory and experiment – will advance our scientific fields.

Consider the next step from information to knowledge. Here one approach is typified by the Semantic Web [6] stressing the use of XML based meta-data to express a large number of linked "information nuggets" from which knowledge comes as an emergent phenomenon. Another vision comes from DoE's ASCI (Accelerated Strategic Computing Initiative) program, which asserts that high fidelity simulations can produce knowledge with modest observational support. More generally we suggest a hallmark of the next decade will be the integration of such simulations with the data deluge. Here data assimilation (common already in weather and other fields) closely integrating time dependent simulation and observation can be expected to increase in importance.

2 Key Technology Concepts for e-Science

We can view the structure of our e-Science environment as shown in figs. 2 and 3 [7]. There are some "real things" (users, computers, instruments), which we term external resources – these are the outer band around the "middleware egg". As shown in fig. 3, these are linked by a collection of "Web Services"[8]. All entities (external resources) are linked by messages whose communication forms a distributed system integrating the component parts. This architectural vision is not new but features of its implementation and its breadth are. This architecture is

similar to that designed commercially for applications like e-commerce and online gaming but is enhanced with many capabilities developed by the High Performance Computing and Communication community. The architecture at the high level of figures 1 2 and 3 is rather trivial but there are some new key and profound ideas.
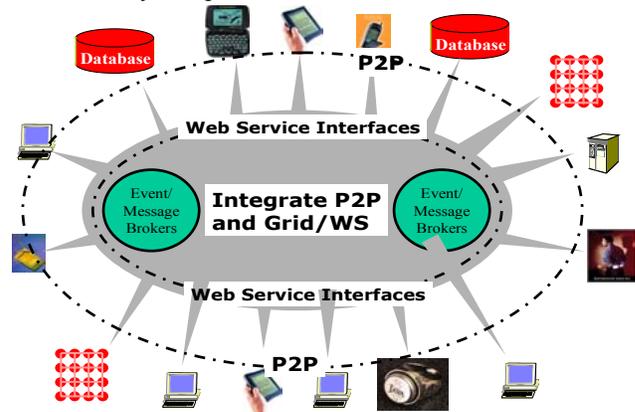

Fig. 2: A Peer-to-Peer Grid

Distributed Object technology is implemented with objects defined in a XML-based IDL (Interface Definition Language) called WSDL (Web Services Definition Language). This allows "traditional approaches" like CORBA or Java to be used "under-the-hood" with an XML wrapper providing a uniform interface. All major companies have endorsed this approach with Microsoft developing .NET and IBM, Oracle, Sun and others preferring a Java oriented approach.
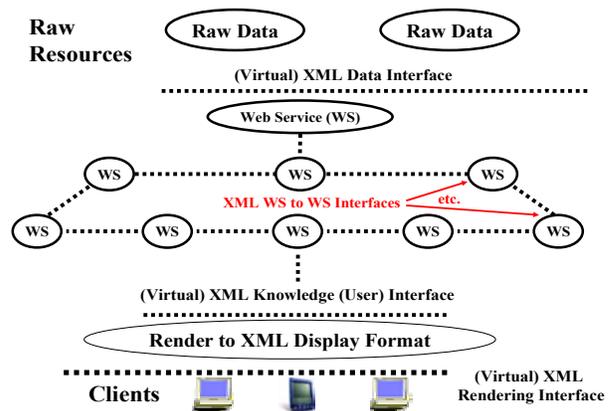

Fig. 3: Role of Web Services (WS) and XML in linkage of clients and raw resources

Another key concept – that of the resource – comes from the web consortium W3C. Everything – whether an external or internal entity – is a resource labeled by a URI (Universal Resource Identifier); a typical form being escience://myplace/mything/mypropertygroup/leaf. This includes not only macroscopic constructs like computer programs or sensors but also their detailed properties. One can consider the URI as the barcode of the Internet – it

labels everything. There are also of course URL's (Universal Resource Locations) which tell you where things are. One can equate these concepts (URI and URL) but this is in principle inadvisable.

Finally our e-science environment is built in a service model. A service is an entity that accepts one or more inputs and gives one or more results. These inputs and results are the messages that characterize the system. In WSDL, the inputs and outputs are termed ports and WSDL defines an overall structure for the messages. The e-Science environment is built in terms of the composition of services.

In summary everything is a resource. The basic macroscopic entities exposed directly to users and to other services are built as distributed objects that are constructed as services so that capabilities and properties are accessed by a message-based protocol. Services contain multiple properties, which are themselves individual resources.

The above is a natural generalization of conventional approaches. A service corresponds roughly to a computer program or process; the ports (interface of a communication channel with a Web Service) to subroutine calls with input parameters and returned data. The critical difference from the past is that one assumes that each service runs on a different computer scattered around the globe. Typically services can be dynamically migrated between computers. Distributed object technology allows us to properly encapsulate the services and provide a management structure. The use of XML and standard interfaces like WSDL give a critical universality that allows the interoperability of services from different sources.

There are several important technology research and development areas on which the e-Science infrastructure builds

1) Basic system capabilities packaged as Web Services. These include security, access to computers (job submittal, status etc.) and access to various forms of databases (information services) including relational systems, LDAP, and XML databases/files. Network wide search techniques about web services or the content of web services could be included here.

2) The messaging sub-system between Web Services and external resources addressing functionality, performance and fault-tolerance.

3) Tool-kits to enable applications to be packaged as Web Services and construction of "libraries' or more precisely components. Near-term targets include areas like image processing used in virtual observatory projects or gene searching used in bio-informatics.

4) Application meta-data needed to describe all stages of the scientific endeavor.

5) Higher-level and value-added system services such as network monitoring, collaboration and visualization.

6) What has been called the Semantic grid or approaches to the representation of and discovery of knowledge from Grid resources. This was briefly discussed at the end of section 1.

7) Portal technology defining user facing ports on web services which accept user control and deliver user interfaces.

Fig. 3 is drawn as a classic 3-tier architecture: client (at the bottom), back-end resource (at the top) and multiple layers of middleware (constructed as web services). This is the natural virtual machine seen by a given user accessing a resource. However the implementation could be very different. Access to services can be mediated by "servers in the core" or alternatively by direct peer-to-peer (P2P) interactions between machines "on the edge". The distributed object abstractions with separate service and message layers allow either P2P or server-based implementations. The relative performance of each approach (which could reflect computer/network horsepower as well as existence of firewalls) would be used in deciding on the implementation to use. P2P approaches best support local dynamic interactions; the server approach scales best globally but cannot easily manage the rich structure of transient services, which would characterize complex tasks. We refer to our architecture as a peer-to-peer grid with peer groups managed locally arranged into a global system supported by core servers.

3 Characteristics of e-Science Infrastructure

We can ask if this new approach to the science infrastructure affects key hardware, software infrastructure and their performance requirements. First we present some general remarks. Servers tend to be highly reliable these days. Typically they run in controlled environments but also their software can be proactively configured to ensure reliable operation. One can expect servers to run for months on end and often one can ensure that they are modern hardware configured for the job at hand. Clients on the other hand can be quite erratic with unexpected crashes and network disconnections as well as sporadic connection typical of portable devices. Transient material can be stored on clients but permanent information repositories must be on servers – here we talk about "logical" servers as we may implement a session entirely within a local peer group of "clients". Robustness of servers needs to be addressed in a dynamic fashion and on a scale greater than in previous systems. However traditional techniques – replication and careful transaction processing – probably can be extended to handle servers and the web services that they host. Clients realistically must be assumed to be both unreliable and sort of outside our control. Some clients will be "antiques" and underpowered and are likely to have many software hardware and network instabilities. In the simplest model clients "just" act as a vehicle to render information for the user with all the action on "reliable" servers. Here applications like Microsoft Word "should be" packaged as Web services with message based input and output. Of course if you have a wonderful robust PC you can run both server(s) and thin client on this system.

Finally we turn to the communication subsystem, which has very interesting characteristics of a Jekyll and

Hyde nature. Examining the growing power of optical networks we see the increasing universal bandwidth that in fact motivates the thin client and server based application model. However the real world also shows slow networks (such as dial-ups), links leading to a high fraction of dropped packets and firewalls stopping our elegant application channels dead in their tracks. We also see some chaos today in the telecom industry which is stunting somewhat the rapid deployment of modern "wired' (optical) and wireless networks. We suggest that key to future e-Science infrastructure will be messaging subsystems that manage the communication between external resources, web services and clients to achieve the highest possible system performance and reliability. We suggest this problem is sufficiently hard that we only need solve this problem "once" i.e. that all communication – whether TCP/IP, UDP, RTP, RMI, XML or you-name-it be handled by a single messaging or event subsystem. Note this implies we would tend to separate control and high volume data transfer reserving specialized protocols for the latter and more flexible robust approaches for setting up the control channels. In the next section we discuss the architecture of an e-Science message service and in the final section 5, discuss the performance of a particular system Narada that we built.

4 Architecture of an Event Service

Now we examine a possible approach to handling the communication infrastructure discussed at the end of the
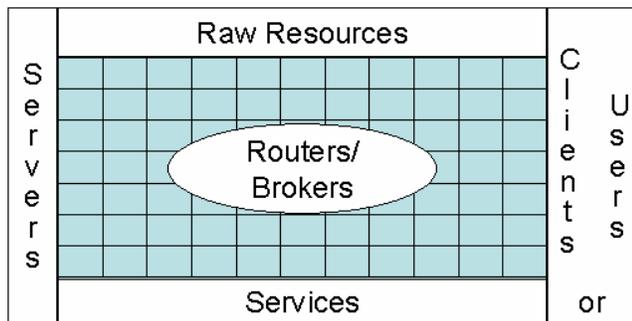

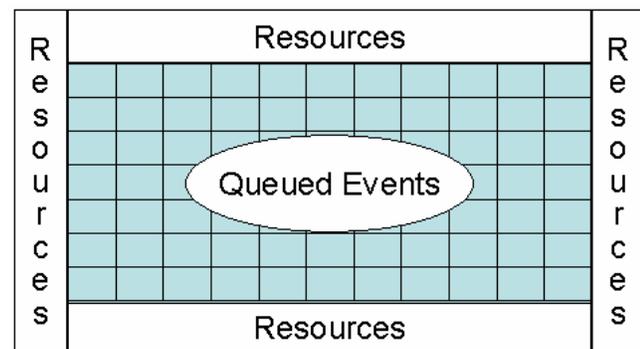*Fig. 4: One View of System Components with event service represented by central mesh*

Fig. 5: Simplest View of System Components showing routers/brokers of event service supporting queues

last section. As shown in fig. 4, we see the event service as linking all parts of the system together and this can be simplified further as in fig. 5 – the event service is to provide the communication infrastructure needed to link resources together. We show this less abstractly in fig. 6.There are routers or brokers whose function is to distribute messages between the raw resources, clients and servers of the system. We consider that the servers provide services (perhaps defined in the WSDL [8] and related XML standards [9]) and do NOT distinguish at this level between what is provided (a service) and what is providing it (a server). Note that we do not distinguish between events and messages; an event is defined by some XML Schema including a time-stamp but the latter can of course be absent to allow a simple message to be thought of as an event. Note an event is itself a resource and might be archived in a database raw resource. Routers and brokers actually provide a service – the management of (queued events) and so these can themselves be considered as the servers corresponding to the event or message service. This will be discussed a little later as shown in fig. 7. Here we note that we design our event systems to support some variant of the publish-subscribe mechanism. Messages are queued from "publishers" and then clients subscribe to them. XML tag values are used to define the "topics" or "properties" that label the queues.
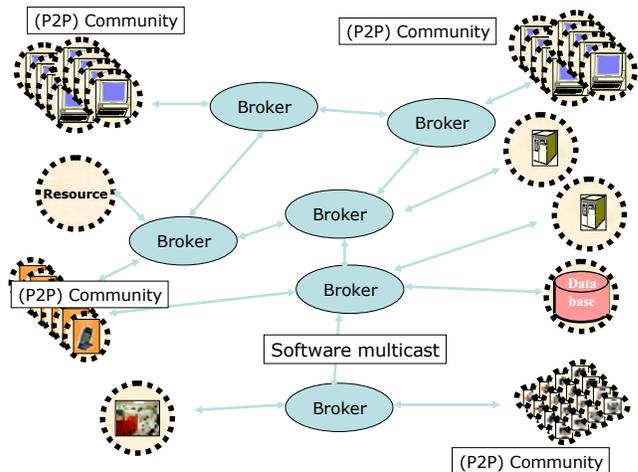

Fig. 6: Distributed Brokers Implementing Event Service

Note that in fig. 3, we call the XML Interfaces "virtual". This signifies that the interface is logically defined by an XML Schema but could in fact be implemented differently. As a trivial example, one might use a different syntax with say *<sender>meoryou</sender>* replaced by *sender:meoryou* which is an easier to parse but less powerful notation. Such simpler syntax seems a good idea for "flat" Schemas that can be mapped into it. Less trivially, we could define a linear algebra web service in WSDL but compile it into method calls to a Scalapack routine for high performance implementation. This compilation step would replace the XML SOAP based messaging [9] with serialized method arguments of the default remote invocation of this

service by the natural in memory stack based use of pointers to binary representations of the arguments.

We build on several interesting recent developments in the messaging area and we quote three examples: there is SOAP messaging [9]; the JXTA peer-to-peer protocols [10]; the commercial JMS message service [11]. All these approaches define messaging principles but not always at the same level of the OSI stack; further they have features that sometimes can be compared but often they make implicit architecture and implementation assumptions that hamper interoperability and functionality. We suggest breaking such frameworks into subsystem capabilities describing common core primitives. This will allow us to compose them into flexible systems, which support a range of functionality without major change in application interfaces. Here SOAP defines a message structure and is already a "core primitive" as described above; it is "only" XML but as discussed above, a message specified in XML could be "compiled to other forms such as RMI either for higher performance or "just" because the message was linking two Java programs. Note that we like publish-subscribe messaging mechanisms but this is sometimes unnecessary and indeed occurs unacceptable overhead. We term the message queues in figs. 5 and 7 virtual to indicate that the implicit publish-subscribe mechanism can be bypassed if this agreed in initial negotiation of communication channel.
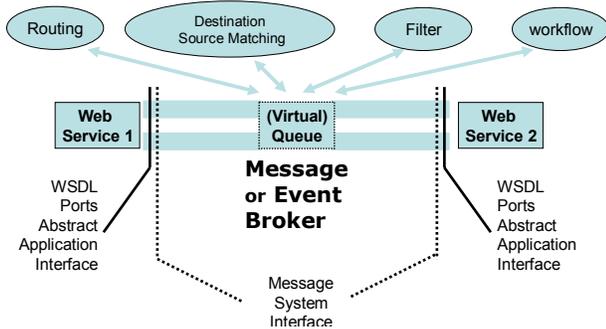

Fig 7: Communication Model showing Sub-services of Event Service

However it does appears useful to define an event architecture such as that of fig. 7, allowing communication channels between Web services which can either be direct or pass through some mechanism allowing various services on the events. These could be low-level such as routing between known source and destination or the higher-level publish-subscribe mechanism that identifies the destinations for a given published event. Some routing mechanisms in peer-to-peer systems in fact use dynamic strategies that merge these high and low level approaches to communication. Note that the messages must support multiple interfaces: as a "physical" message it should support SOAP; above this it support added services such as filtering, publish-subscribe, collaboration, workflow which correspond to changing message content or delivery. Above this there are application and service standards. All of these

are defined in XML, which can be virtualized. As an example, consider an audio-video conferencing web service [13]. It could use a simple publish/subscribe mechanism to advertise the availability of some video feed. A client interested in receiving the video would negotiate (using the SIP protocol perhaps) the transmission details. The video could either be sent directly from publisher to subscriber; alternatively from publisher to web service and then from web service to subscriber; as a third option, we could send from the web service to the client but passing through a filter that converted one codec into another if required. In the last case, the location of the filter would be negotiated based on computer/network performance issues – it might also involve proprietary software only available at special locations. The choice and details of these three different video transport and filtering strategies would be chosen at the initial negotiation and one would at this stage "compile" a generic interface to its chosen form. One could of course allow dynamic "run-time compilation" when the event processing strategy needs to change during a particular stream. This scenario is not meant to be innovative but rather to illustrate the purpose of our architecture building blocks in a homely example. Web services are particularly attractive due to their support of interoperability, which allows the choices described.

We have designed and implemented a system Narada supporting the described here with a dynamic collection of brokers supporting a generalized publish-subscribe mechanism. As described elsewhere [14-16] this can operate either in a client-server mode like JMS or in a completely distributed JXTA-like peer-to-peer mode. By combining these two disparate models, Narada can allow optimized performance-functionality trade-offs for different scenarios. Note that typical overheads for broker processing are around a millisecond. This is unacceptable for applications like MPI for parallel processing, which needs microsecond latency but acceptable for real-time collaboration [7,17] and even audio-video conferencing where each frame takes around 30 milliseconds. In the following section, we discuss our initial performance results with Narada,

5 Performance of the Narada System
5.1 Comparison of Narada and SonicMQ:
To gather performance data, we run an instance of the SonicMQ [18] (version 3.0) JMS broker and Narada broker on the same dual CPU (Pentium-3, 1 GHz, 256MB) machine. We then setup 100 subscribers over 10 different JMS Topic Connections on another dual CPU (Pentium-3, 866MHz, 256MB) machine. In addition there is a *measuring* subscriber and a *publisher* that are set up on a third dual CPU (Pentium 3, 866MHz, 256MB RAM) machine. Since we will be computing communication delays setting up the measuring subscriber and publisher on the same machine enables us to obviate the need for clock synchronizations and differing clock drifts. The three machines involved in the benchmarking process have Linux (version 2.2.16) as

their operating system. The runtime environment for the broker, publisher and subscriber processes is Java 2 JRE (Java-1.3.1, Blackdown-FCS, mixed mode). Subscribers subscribe to a certain topic and the publisher publishes to the same topic. Once the publisher starts issuing messages the factor that we are most interested in is the *transit delay* in the receipt of these messages at the subscribers. This delay corresponds to the response times experienced at each of the subscribers. We measure this delay at the measuring subscriber while varying the publish rates and message sizes of the messages being published. We control the publish rates by varying the time interval between the publishing of two consecutive messages. We vary the message size by changing the payload contained in

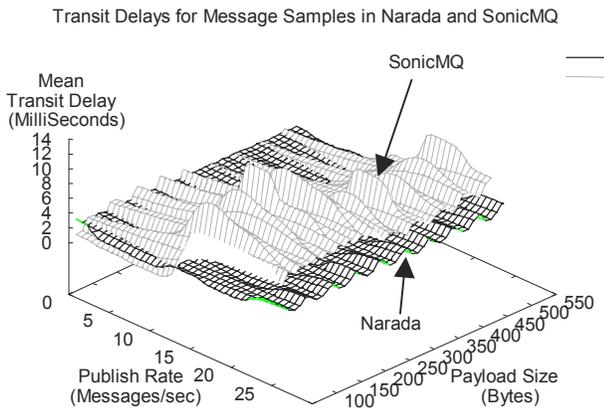Transit Delays for Message Samples in Narada and SonicMQ

Fig. 8: Transit Delays - Lower publish rates and smaller payloads

the message. For a sample of messages received at the measuring subscriber we calculate the *mean transit delay*.

Figures 8-10 depict the transit delays for JMS clients under Narada and SonicMQ for varying publish rates and payload sizes. As can be seen from the results Narada compares very well with SonicMQ while also outperforming SonicMQ in several cases. Furthermore, the standard deviation associated with the message samples (for individual test cases) received at clients in Narada were, for the most part, lower than those at clients in SonicMQ for the cases that were benchmarked.

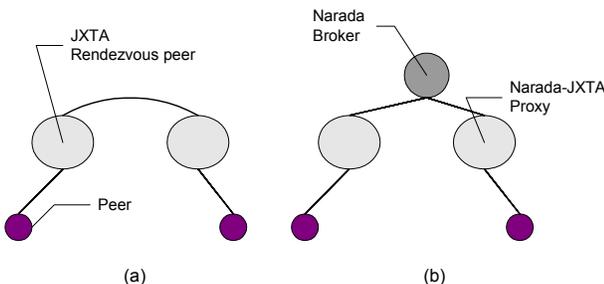## 5.2 Comparison of Narada-JXTA and Pure JXTA:
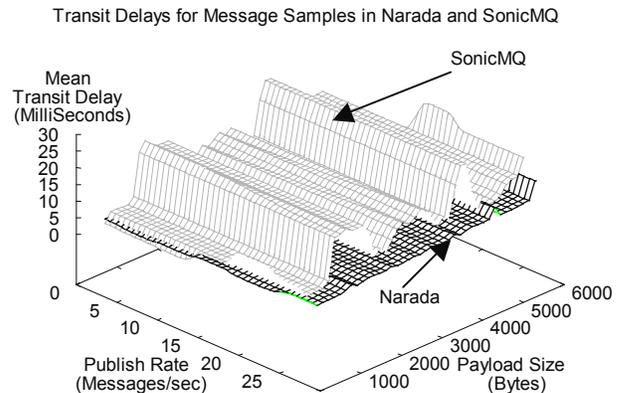

Fig 11: JXTA Narada Benchmark Set-up

Transit Delays for Message Samples in Narada and SonicMQ

Fig. 9: Transit Delays - Lower publish rates and bigger payloads

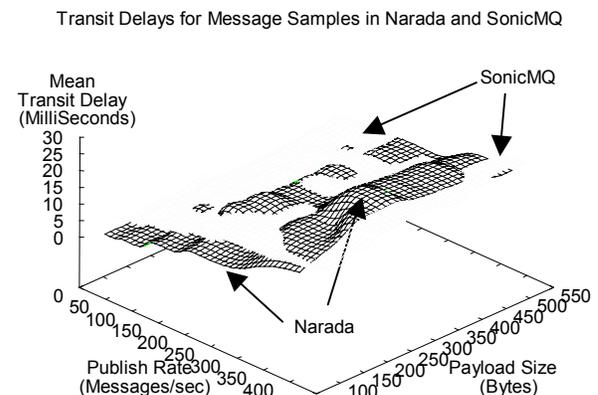
Transit Delays for Message Samples in Narada and SonicMQ

Figure 10: Transit Delays - Higher publish rates and smaller payloads

For comparing JXTA performance in Narada we have the setup depicted in fig. 11. We compare the performance of a pure JXTA environment with the integrated Narada-JXTA system. To compute delays while obviating need for clock synchronizations and the need to account for clock drifts, the two peers were setup on the same machine. The benchmark environment for the pure JXTA case in depicted in fig. 11(a), the two rendezvous peers are connected to each other. The environment for the Narada-JXTA system, fig. 11(b), includes a Narada broker hosted on another machine, the proxies are not connected to each other and are instead connected to the Narada broker. There is another case that we measure and that is the JXTA Direct-P2P case where two peers communicate directly with each other. In all three cases messages published by one of the peers are received at the second peer and the delay is computed. For a given message payload this is done for a sample of messages and we compute the mean delay and the standard deviation associated with the sample. This repeated for different payload sizes.
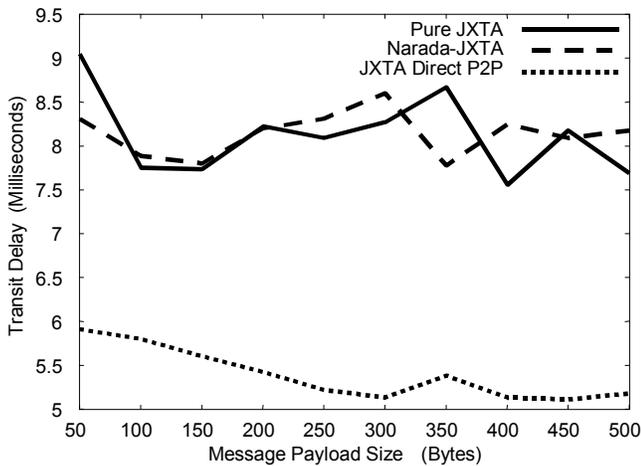
Fig. 12: Comparing Pure JXTA, Narada-JXTA and
Direct P2P for lower payload sizes

Figures 12 and 13 depict the results of our measurements. As can been seen the integrated Narada-JXTA system compares very well with the JXTA system despite the additional hop that is required in the Narada-JXTA case. The Narada-JXTA system also compares quite well with the Direct-P2P case, which involves direct communications between the two peers. Peers however have a restricted set of active connections that they can initiate after which communication delays can add up significantly. Our tests are in our local laboratory environment and do not exploit the advantage of Narada of supporting long distance hops. It is our conjecture that the integrated Narada system will systematically outperform JMS, the Direct-P2P and Pure–JXTA in the cases that mix local and long distance communication. We are currently setting up these performance measurements.
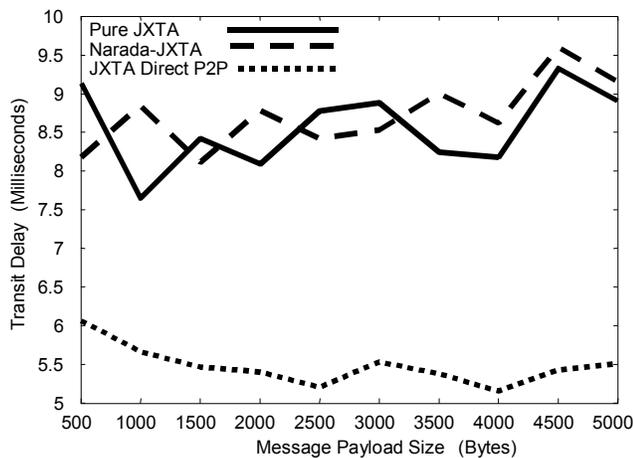

Fig. 13: Comparing Pure JXTA, Narada-JXTA and
Direct P2P for higher payload sizes

Finally we illustrate role of filtering service of our event system. This is used to customize output produced in a collaborative environment involving both conventional desktop clients and PDA's (personal digital assistants) which have limited display characteristics [19], We have

developed an optimized message passing service GMSME (Grid Message Service Micro Edition using our HHMP – the hand-held message protocol) for PDA's shown in fig. 14. Our initial comparison with the SonicMQ (termed GMS in figs 14 and 15) implementation of our collaborative environment is shown in fig. 15. The PDA is a Compaq iPAQ using 802.11 wireless connection but our results will be extended to other systems. Figure 15 shows the average message throughput per second for different message sizes. It is clear that GMSME is significantly slower than GMS
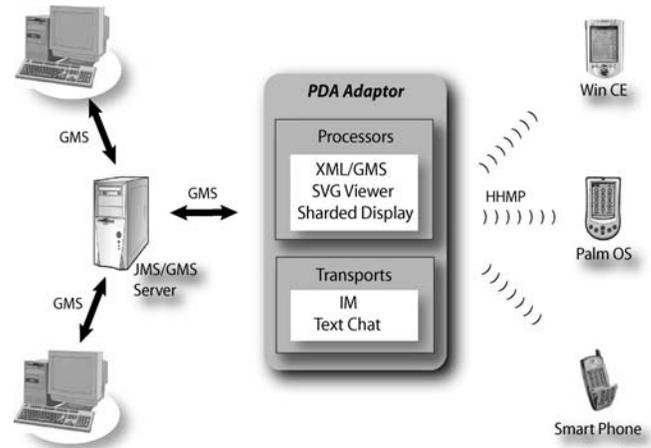

Fig. 14 Event service (GMS) supporting collaboration
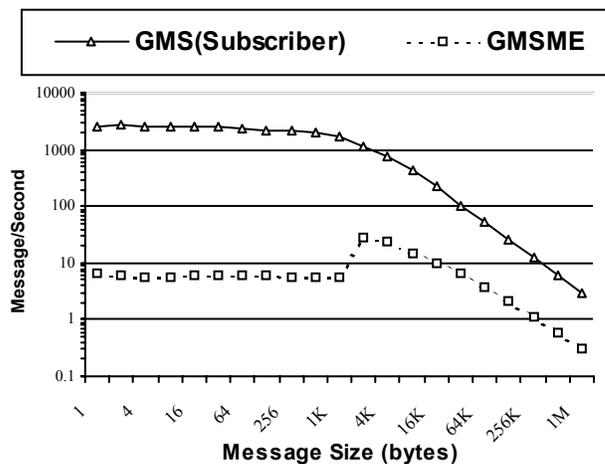between PDA's and desktops

Fig. 15 Comparison of GMSME and GMS for real-time
collaboration for events defining collaboration
synchronization

messages. This is due to low bandwidth of wireless network, synchronization overhead and the slower processing of PDA. In our benchmark, GMSME reaches maximum throughput when message size is 4K and we discuss the implications of this in [19].

References

[1] The Grid Forum http://www.gridforum.org

[2] Globus Grid Project http://www.globus.org

[3] NSF Advisory Committee for Cyberinfrastructure http://www.cise.nsf.gov/b_ribbon/

[4] United Kingdom e-Science Activity http://www.escience-grid.org.uk/

[5] National Virtual Observatory http://www.us-vo.org/

[6] W3C Semantic Web http://www.w3.org/2001/sw/

[7] Geoffrey Fox, Ozgur Balsoy, Shrideep Pallickara, Ahmet Uyar, Dennis Gannon, and Aleksander Slominski, "Community Grids" invited talk at *The 2002 International Conference on Computational Science*, April 21 -- 24, 2002 Amsterdam, The Netherlands

[8] Web Services Description Language(WSDL) version 1.1 http://www.w3.org/TR/wsdl

[9] XML based messaging and protocol specifications SOAP. http://www.w3.org/2000/xp/.

[10] Sun Microsystems JXTA Peer to Peer technology. http://www.jxta.org.

[11] Sun Microsystems. Java Message Service. http://java.sun.com/products/jms.

[12] "Peer-To-Peer: Harnessing the Benefits of a Disruptive Technology", edited by Andy Oram, O'Reilly Press March 2001.

[13] Geoffrey Fox, Wenjun Wu, Ahmet Uyar, Hasan Bulut "A Web Services Framework for Collaboration and Audio/Videoconferencing", to be published in *2002 International Conference on Internet Computing IC'02*: Las Vegas, USA, June 24-27, 2002.

[14] Geoffrey Fox and Shrideep Pallickara "The Narada Event Brokering System: Overview and Extensions" to appear in the proceedings of the *2002 International Conference on Parallel and Distributed Processing Techniques and Applications* (PDPTA'02). http://grids.ucs.indiana.edu/ptliupages/projects/narada/papers/NaradaBrokeringSystem.pdf

[15] Geoffrey Fox and Shrideep Pallickara "JMS Compliance in the Narada Event Brokering System" to appear in the proceedings of the *2002 International Conference on Internet Computing* (IC-02). http://grids.ucs.indiana.edu/ptliupages/projects/narada/papers/JMSSupportInNarada.pdf

[16] Geoffrey Fox and Shrideep Pallickara "Support for Peer-to-Peer Interactions in Web Brokering Systems" to appear in *ACM Ubiquity*. http://grids.ucs.indiana.edu/ptliupages/projects/narada/papers/P2PSystems.pdf

[17] Geoffrey Fox, Marlon Pierce et al., *Grid Services for Earthquake Science*, to be published in Concurrency and Computation: Practice and Experience in ACES Special Issue, Spring 2002. http://aspen.ucs.indiana.edu/gemmauisummer2001/resources/gemandit7.doc

[18] Sonic MQ JMS Broker http://www.sonicsoftware.com/

[19] Geoffrey Fox, Sung-Hoon Ko,Kangseok Kim, Sangyoon Oh, Sangmi Lee, "Integration of Hand-Held Devices into Collaborative Environments" to appear in the proceedings of the *2002 International Conference on Internet Computing* (IC-02). June 24-27 Las Vegas.