Messaging Systems: Parallel Computing the Internet and the Grid

Geoffrey Fox

Indiana University
Computer Science, Informatics and Physics
Community Grids Computing Laboratory,
501 N Morton Suite 224, Bloomington IN 47404
gcf@indiana.edu

Abstract. We contrast the requirements and performance of messaging systems in parallel and distributed systems emphasizing the importance of the five orders of magnitude difference in network hardware latencies in the two cases. We note the importance of messaging in Grid and Web service applications in building the integrated system and the architectural advantages of a message based compared to a connection based approach. We illustrate these points using the NaradaBrokering system and its application to Audio-Video conferencing.

1: Message Passing in Parallel Computing

Parallel Computing has always understood the importance of message passing and PVM and MPI (the topics of this meeting) have dominated this field with other approaches readily mapped into these two systems. The appropriate programming model for parallel systems is of course a very active area with continued research on different architectures (openMP) and different high level approaches involving both domain specific systems and degrees of compiler generated parallelism. The issue has been further invigorated by the successes of the Japanese Earth Simulator system. However even when we use a high level model for parallel programming, message passing is typically essential as the low level primitive ("machine language") for parallelism between distributed memories. In understanding the requirements of message passing, it is useful to divide multi-processor (distributed memory) systems into three classes.

1) Classic massively parallel processor systems (MPP) with low latency high bandwidth specialized networks. One aims at message latencies of one to a few microseconds and scalable bisection bandwidth. Ignoring latency, the time to communicate a word between two nodes should be a modest multiple (perhaps 20) of time taken to calculate a floating point result. This communication performance should be independent of number of nodes in system.

2) Commodity clusters with high performance but non optimized communication networks. Latencies can be in the 100-1000 microsecond range typical of simple socket based communication interfaces.

3) Distributed or Grid systems with possibly very high internode bandwidth but the latency is typically 100 milliseconds or more as familiar from internet travel times.

Of course there is really a spectrum of systems with cost-performance increasing by a factor of 4 or so as one goes from 1) to 3). Here we will focus on the endpoints 1) and 3) – MPP's and the Grid and not worry about intermediate cases like 2). MPI especially is aimed at the class 1) with optimized "native" implementations exploiting particular features of the network hardware. Generic versions of PVM and MPI using socket based communication on a localized network illustrate 2) as do many other specialized programming environments (such as agent-based approaches). The latter typically cannot afford the development effort to optimize communication and as illustrated by our latter discussion of Grid messaging requires substantially more functionality than MPI and PVM. Grid systems are very diverse and there is little understanding at this stage as to critical performance and architecture (topology) characteristics. As we discuss in sections 2 and 3, they need a rich messaging environment very different from MPI and PVM. Note this doesn't mean that we shouldn't port systems like MPI to the Grid as in MPICH-G2 [1] and PACX-MPI [2]; there is clearly a need to run all messaging environments on all classes of machine.

The overriding "idea" of this paper is that messaging for an application with intrinsic (hardware) latency L, mustn't have software and routing overheads greater than this but certainly can afford extra software overheads of size around 0.1L without "noticing it". This implies that it should be expected that the application classes 1) 2) 3) have very different messaging semantics. MPI and PVM are not totally "barebones" but they are optimized for fast message processing and little communication overhead due to headers in the message packets.

Parallel computing can usually use very lean messaging as one is sending between different parts of the "same" computation; thus the messages can usually just contain data and assume that the recipient process understands the context in which the data should be interpreted.

2: Messaging in Grids and Peer-to-Peer Networks

Now let us consider messaging for the Grid and peer-to-peer (P2P) networks which we view as essentially identical concepts [3]. Here we are not given a single large scale simulation – the archetypical parallel computing application, Rather ab initio we start with a set of distributed entities – sensors, people, codes, computers, data archives – and the task is to integrate them together. For parallel computing one is decomposing applications into parts and messaging reflects that the parts are from the same whole. In distributed computing, the initial entities are often quite distinct and it is the messaging that links them together. Correspondingly the messaging for the Grid must carry the integration context and not just data; thus one has both the

time (typically the 100 millisecond network latency) and the need for a much richer messaging system on the Grid than for parallel computing.

In parallel computing explicit message passing is a necessary evil as we haven't found a generally applicable high level expression of parallelism.. For Grids and P2P networks, messaging is the natural universal architecture which expresses the function of the system. In the next sections we compare the requirements for a messaging service in the two cases.

2.1: Objects and Messaging

Object-based programming models are powerful and should be very important in scientific computing even though up to now both C++ and Java have not achieved widespread use in the community [5]. The natural objects are items like the collection of physical quantities at a mesh point or at a larger grain size the arrays of such mesh points [6, 7]. There is some overhead attached with these abstractions but there are such natural objects for most parallel computing problems. However one can also consider the objects formed by the decomposed parts of a parallel application – it has *not* been very helpful to think of the decomposed parts of parallel applications as objects for these are not especially natural components in the system; they are what you get by dividing the problem by the number of processors. On the other hand, the linked parts in a distributed system (Web, Grid, P2P network) are usefully thought of objects as here the problem creates them; in contrast they are created for parallel computing by adapting the problem to the machine architecture. The Grid distributed objects are nowadays typically thought of as Web services and we will assume this below. We will also not distinguish between objects and services. Note that objects naturally communicate by messages linking the exposed interfaces (remote procedure calls or ports) of the distributed objects. So Grid messaging is the natural method to integrate or compose objects (services); parallel computing messaging is the natural representation of the hardware – not the application.

2.2 Requirements for a Grid Messaging Service

There are common features of messaging for distributed and parallel computing; for instance messages have in each case a source and destination. In P2P networks especially, the destination may be specified indirectly and determined dynamically while the message is en route using properties (published meta-data) of the message matched to subscription interest from potential recipients. Groups of potential recipients are defined in both JXTA [8] for P2P and MPI for parallel computing. Publish-subscribe is a particularly powerful way to dynamically define groups of message recipients. Collective communication – messages sent by hardware or software multicast – is important in all cases; much of the complexity of MPI is devoted to this. Again one needs to support in both cases, messages containing complex data struc-

tures with a mix of information of different types. One must also support various synchronization constraints between sender and receiver; messages must be acknowledged perhaps. These general characteristics are shared across messaging systems. There are also many differences where perhaps as discussed in section 1, performance is perhaps the most important issue.

Now consider message passing for a distributed system. Here we have elegant objects exchanging messages that are themselves objects. It is now becoming very popular to use XML for defining the objects and messages of distributed systems. Fig. 1 shows our simple view of a distributed system – a Grid or P2P Network – as a set of XML specified resources linked by a set of XML specified messages. A resource is any entity with an electronic signature; computer, database, program, user, sensor.

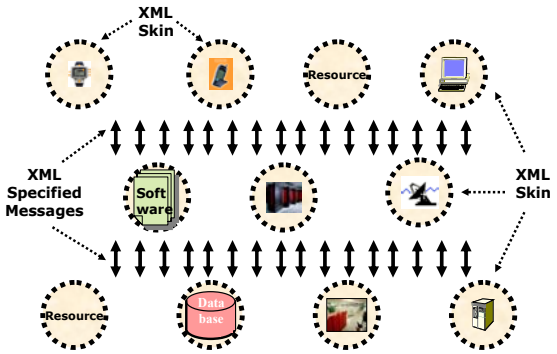

Fig. 1: XML Specified Resources linked by XML Specified Messages

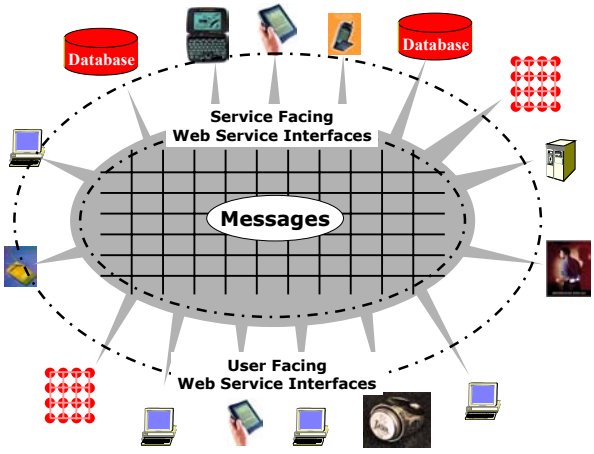

Fig. 2: A Peer-to-Peer Grid constructed from Web Services with both user-facing and service-facing ports to send and receive messages

The web community has introduced SOAP [9] which is essentially the XML message format postulated above and "Web services" which are XML specified distributed objects. Web services are "just" computer programs running on one of the computers in our distributed set. Web services send and receive messages on so-called ports – each port is roughly equivalent to a subroutine or method call in the "old programming model". The messages define the name of the subroutine and its input and if necessary output parameters. This message interface is called WSDL (Web Service Definition Lan-

guage [10]) and this standard is an important W3C consortium activity. Using Web services for the Grid requires extensions to WSDL and the resultant OGSI [11] and OGSA (Open Grid Service Architecture [12]) standards are major efforts in the Grid forum [13] at the moment. OGSI is the component model and OGSA the interface standards that Grid services and messages must respect.

As seen in the peer-to-peer Grid of fig. 2, ports are either user-facing (messages go between user and Web Services) or service or resource-facing where messages are exchanged between different Web services. As discussed in [14] there is a special variant of WSDL for user-facing ports – WSRP (Web Services for Remote Portlets [15]) which defines a component model for user interfaces. This is one example of the context carried by Grid messages – WSRP indicates a user interface message that can be processed by aggregation portals like Apache Jetspeed [16].

One particularly clever idea in WSDL is the concept that one first defines not methods themselves but their abstract specification. Then there is part of WSDL that "binds" the abstract specification to a particular implementation. Here one can choose to bind the message transport not to the default HTTP protocol but to a different and perhaps higher performance protocol. For instance if one had ports linking Web services on the same computer, then these could in principle be bound to direct subroutine calls. This concept has interesting implications for building systems defined largely in XML at the level of both data structure and methods. Further one can imagine some nifty new branch of compilation which automatically converted XML calls on high performance ports and generated the best possible implementation.

2.3: Performance of Grid Messaging Systems

Now let us discuss the performance of the Grid messaging system. As discussed in section 1, the Grid messaging latency is very different from that for MPI as it can take several 100 milliseconds for data to travel between two geographically distributed Grid nodes; in fact the transit time becomes seconds if one must communicate between the nodes via a geosynchronous satellite. One deduction from this is that the Grid is often not a good environment for traditional parallel computing. Grids are not dealing with the fine grain synchronization needed in parallel computing that requires the few microsecond latency seen in MPI for MPP's. For us here, another more interesting deduction is that very different messaging strategies can be used in Grid compared to parallel computing. In particular we can perhaps afford to invoke an XML parser for the message and in general invoke high level processing of the message. Here we note that interspersing a filter in a message stream – a Web service or CORBA broker perhaps – increases the transit time of a message by about 0.5 millisecond; small compared to typical Internet transit times. This allows us to consider building Grid messaging systems which have substantially higher functionality than traditional parallel computing systems. The maximum acceptable latency is application dependent. Perhaps one is doing relatively tightly synchronized computations among multiple Grid nodes; the high latency is perhaps hidden by overlapping communication and computation. Here one needs tight control over the latency and re-

duce it as much as possible. On the other extreme, if the computations are largely independent or pipelined, one only needs to ensure that message latency is small compared to total execution time on each node. Another estimate comes from audio-video conferencing [17]. Here a typical timescale is 30 milliseconds – the time for a single frame of video conferencing or a high quality streaming movie. This 30 ms. scale is not really a limit on the latency but in its variation or jitter shown later in fig. 4. In most cases, a more or less constant offset (latency) can be tolerated.

Now consider, the bandwidth required for Grid messaging. Here the situation is rather different for there are cases where large amounts of information need to be transferred between Grid nodes and one needs the highest performance allowed by the Network. In particular numbers often need to be transferred in efficient binary form (say 64 bits each) and not in some XML syntax like <number>3.14159</number> with 24 characters requiring more bandwidth and substantial processing overhead. There is a simple but important strategy here and now we note that in fig. 1, we emphasized that the messages were specified in XML. This was to allow one to implement the messages in a different fashion which could be the very highest performance protocol. As explained above, this is termed binding the ports to a particular protocol in the Web service WSDL specification. So what do we have left if we throw away XML for the implementation? We certainly have a human readable interoperable interface specification but there is more which we can illustrate again by audio-video conferencing, which is straight-forward to implement as a Web service [18]. Here A/V sessions require some tricky set-up process where the clients interested in participating, join and negotiate the session details. This part of the process has no significant performance issues and can be implemented with XML-based messages. The actual audio and video traffic does have performance demands and here one can use existing fast protocols such as RTP. This is quite general; many applications need many control messages, which can be implemented in basic Web service fashion and just part of the messaging needs good performance. Thus one ends up with control ports running basic WSDL with possible high performance ports bound to a different protocol.

3: Narada Brokering Messaging Services

Shrideep Pallickara in the Community Grids Laboratory at Indiana has developed [19, 20] a message system for Web resources designed according to the principles sketched above. It is designed to be deployed as a hierarchical network of brokers that handle all aspects of Grid and Web distributed systems that can be considered as "only connected to the message". One critical design feature is that one considers the message and not the connection as the key abstraction. Destinations, formats and transport protocols are "virtualized" i.e. specified indirectly by the user at a high level. Messages are labeled by XML topics and used to bind source and destinations with a publish-subscribe mechanism. The transport protocol is chosen using a Network Weather Service [21] like evaluation of the network to satisfy quality of service constraints. A given message can be routed through a (logarithmic) network of Na-

rada brokers using if needed a different protocol at each link. For example, an audio stream might have a TCP/IP link through a firewall followed by a UDP link across a high latency reliable satellite link. Currently there is support for TCP, UDP, Multicast, SSL, raw RTP and specialized PDA clients. Also NaradaBrokering (NB) provides the capability for communication through firewalls and proxies. It can operate either in a client-server mode like JMS (Java Message Service [22]) or in a completely distributed JXTA-like [8] peer-to-peer mode. Some capabilities of importance include

(1) NB Supports heterogeneous network transportation and provides unified multipoint transportation

Software multicast – Since NB relies on software multicast, entities interested in linking collaboratively with each other need not set up a dedicated multicast group for communication. Each NB broker can handle hundreds of clients and can be arranged in general networks. Further as shown in fig. 3, the typical delay on a fast network is

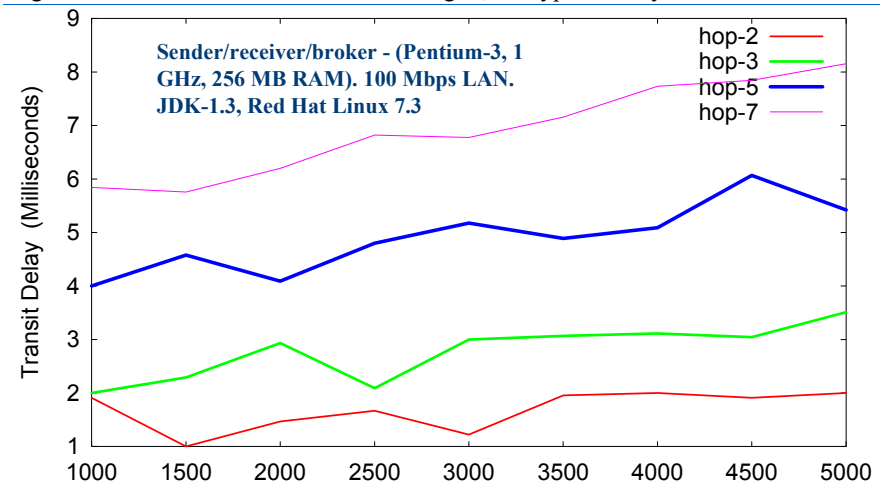

Fig. 3: Transit Delays for NaradaBrokering

less than a millisecond per hop between brokers. Thus software multicast appears practical under general circumstances.

Communication over firewalls and proxy boundaries – NB incorporates strategies to tunnel through firewalls and authenticating proxies such as Microsoft's ISA and those from iPlanet and Checkpoint.

Communication using multiple transport protocols – We described above how this can be effectively used to provide quality of service.

(2) NB provides robust, scalable and high efficient multipoint transportation services

Availability and scalability – There is no single point of failure within the NB messaging system. Additional broker nodes may be added to support large heterogeneous distributed systems. NB's cluster based architecture allows the system to scale. The number of broker nodes may increase geometrically, but the communication *path length*s between nodes increase logarithmically.

Efficient routing and bandwidth utilizations – NB efficiently computes destinations associated with an event. The resultant routing solution chooses links efficiently to reach the desired destinations. The routing solution conserves bandwidth by not overload links with data that should not be routed on them. Under conditions of high loads the benefits accrued from this strategy can be substantial.

Security – NB uses a message-based security mechanism that avoids difficulties with connection (SSL) based schemes and will track the emerging Web service standards in this area [23].

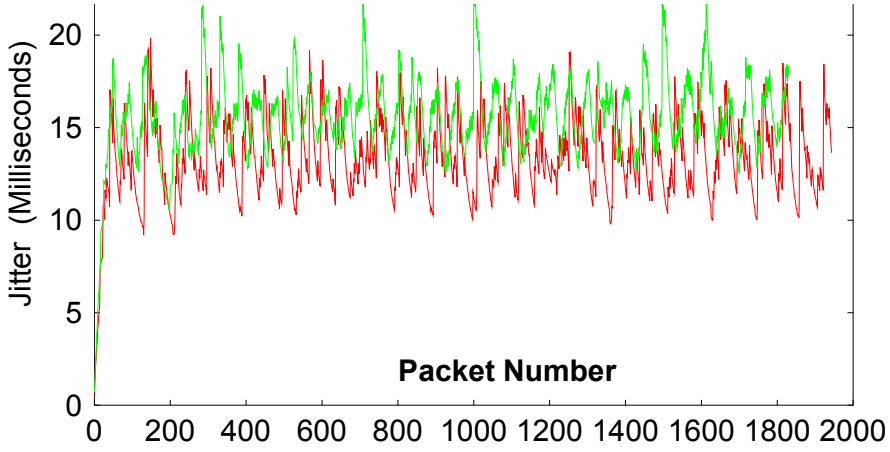

Fig. 4: Jitter (roughly standard deviation) in ms. for a single broker handling 400 video clients with a total bandwidth of 240 Mbps. The lower red lines are for NB using publish-subscribe and RTP transport; the upper green line is for a standard Java Media Framework video server. The mean interval between packets is 30 ms.

Typical performance measurements for NB are given in figures 3 and 4. Further it compares well with the performance of commercial JMS and JXTA implementations. Future work will develop NB to support the emerging Web service messaging standards in areas of addressing [24], reliability [25] and security [23]. One can build Grid hosting environments on NaradaBrokering that allow efficient flexible federation of Grids with different architectures.

References

1. MPICH-G2 grid-enabled implementation of the MPI v1.1 standard based on the MPICH library http://www.nsf-middleware.org/NMIR3/components/mpichg2.asp

2. PACX-MPI described in M. Mueller , E. Gabriel and M. Resch, *A Software Development Environment for Grid Computing*, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15, pages 1543-1552, 2002.
3. Geoffrey Fox, Dennis Gannon, Sung-Hoon Ko, Sangmi Lee, Shrideep Pallickara, Marlon Pierce, Xiaohong Qiu, Xi Rao, Ahmet Uyar, Minjun Wang, Wenjun Wu, *Peer-to-Peer Grids*, Chapter 18 of Reference [4].
4. *Grid Computing: Making the Global Infrastructure a Reality* edited by Fran Berman, Geoffrey Fox and Tony Hey, John Wiley & Sons, Chichester, England, ISBN 0-470-85319-0, March 2003. http://www.grid2002.org
5. High Performance Java http://www.hpjava.org.
6. Zoran Budimlic, Ken Kennedy, and Jeff Piper. *The cost of being object-oriented: A preliminary study*. Scientific Programming, 7(2):87-95, 1999.
7. S. Markidis, G. Lapenta and W.B. VanderHeyden, Parsek: *Object Oriented Particle in Cell Implementation and Performance Issues*. Java Grande Conference 2002 and Concurrency and Computation: Practice and Experience, to be published.
8. Project JXTA Peer-to-peer system http://www.jxta.org/
9. SOAP: Simple Object Access Protocol http://www.w3.org/TR/SOAP/
10. WSDL: Web Services Description Language http://www.w3.org/TR/wsdl.html.
11. OGSI Open Grid Service Infrastructure Working Group of Global Grid Forum http://www.gridforum.org/ogsi-wg/
12. Open Grid Services Architecture (OGSA) http://www.gridforum.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf
13. Global Grid Forum http://www.gridforum.org
14. G. Fox, D. Gannon, M. Pierce, M. Thomas, *Overview of Grid Computing Environments*, Global Grid Forum Informational Document http://www.gridforum.org/documents/
15. OASIS Web Services for Remote Portlets (WSRP) http://www.oasis-open.org/committees/
16. Apache Jetspeed Portal http://jakarta.apache.org/jetspeed/site/index.html
17. Ahmet Uyar, Shrideep Pallickara and Geoffrey Fox *Audio Video Conferencing in Distributed Brokering Systems* in Proceedings of the 2003 International Conference on Communications in Computing, Las Vegas June 2003, http://grids.ucs.indiana.edu/ptliupages/publications/NB-AudioVideo.pdf.
18. Geoffrey Fox, Wenjun Wu, Ahmet Uyar, Hasan Bulut, Shrideep Pallickara, *A Web Services Framework for Collaboration and Videoconferencing*. WACE Conference Seattle June 2003. http://grids.ucs.indiana.edu/ptliupages/publications/finalwacepapermay03.doc.
19. NaradaBrokering from Indiana University http://www.naradabrokering.org
20. Shrideep Pallickara and Geoffrey Fox *NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids* in Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003, Rio Janeiro, Brazil June 2003. http://grids.ucs.indiana.edu/ptliupages/publications/NB-Framework.pdf
21. Network Weather Service NWS http://www.nsf-middleware.org/documentation/NMI-R3/0/NWS/index.htm.
22. JMS: Java Message Service http://java.sun.com/products/jms/.
23. Yan Yan, Yi Huang, Geoffrey Fox, Ali Kaplan, Shrideep Pallickara, Marlon Pierce and Ahmet Topcu, *Implementing a Prototype of the Security Framework for Distributed Brokering Systems* in Proceeedings of 2003 International Conference on Security and Management (SAM'03: June 23-26, 2003, Las Vegas, Nevada, USA, http://grids.ucs.indiana.edu/ptliupages/publications/SecurityPrototype.pdf.
24. Draft Web Service Addressing Standard from IBM and Microsoft http://msdn.microsoft.com/ws/2003/03/ws-addressing/
25. Draft Web Service Reliable Messaging Standard from BEA IBM Microsoft and TIBCO http://www-106.ibm.com/developerworks/library/ws-rm/.