Implementing a NTP-Based Time Service within a Distributed Middleware System

Hasan Bulut, Shrideep Pallickara and Geoffrey Fox
(hbulut, spallick, gcf)@indiana.edu
Community Grids Lab, Indiana University

Abstract: Time ordering of events generated by entities existing within a distributed infrastructure is far more difficult than time ordering of events generated by a group of entities having access to the same underlying clock. Network Time Protocol (NTP) has been developed and provided to the public to let them adjust their local computer time from single (multiple) time source(s), which are usually atomic time servers provided by various organizations, like NIST and USNO. In this paper, we describe the implementation of a NTP based time service used within NaradaBrokering, which is an open source distributed middleware system. We will also provide test results obtained using this time service.
Keywords: Distributed middleware systems, network time services, time based ordering, NTP

1. Introduction

In a distributed messaging system, messages are timestamped before they are issued. Local time is used for timestamping and because of the unsynchronized clocks, the messages (events) generated at different computers cannot be time-ordered at a given destination.

On a computer, there are two types of clock, hardware clock and software clock. Computer timers keep time by using a quartz crystal and a counter. Each time the counter is down to zero, it generates an interrupt, which is also called one clock tick. A software clock updates its timer each time this interrupt is generated. Computer clocks may run at different rates. Since crystals usually do not run at exactly the same frequency two software clocks gradually get out of sync and give different values. The accuracy of computer clocks can vary due to manufacturing defects, changes in temperature, electric and magnetic interference, the age of the oscillator, or computer load [1]. Another issue is that an ill-behaved software program can use the timer's counter and change the interrupt rate. This could cause the clock to rapidly gain or lose time. A software clock looses state when the machine is turned off and it synchronizes itself with the hardware clock at reboot. Furthermore, hardware clock itself may not be synchronized with the real time and may be seconds, minutes or even days off. Finally, a software clock's accuracy would be bounded by the accuracy of the hardware clock at the reboot.

Because of hardware or software reasons, computer clocks may run slower or faster than they should. An ideal clock is a clock whose derivative with respect to real time is equal to 1. If this derivative is smaller than 1 it is considered a slow clock, and if this derivative is greater than 1 it is considered a fast clock.

It is thus necessary to synchronize the clocks in a distributed system, if the time-based order of messages matter. One cannot rely on the underlying hardware or software clocks to provide synchronization. One of the most widely used algorithms in the Internet is the Network Time Protocol (NTP) [2-3], which has been around more than a decade and may provide an accuracy of 1-30 ms range, where accuracy implies that using NTP the underlying clock is within 30 ms of time server clock (usually an atomic clock). NTP achieves this accuracy by using advanced clock synchronization algorithms. There are atomic time servers provided by various organizations, i.e. National Institute of Standards and Technology (NIST) [4] and U.S. Naval Observatory (USNO) [5]. NIST and USNO Internet Time Services use

multiple *stratum-1*[1] time servers, which are open to public access. Using a NTP client, anyone can synchronize their clock with the atomic time server time within that range.

NaradaBrokering [6 -10] messaging system has also been used to carry audio/video (A/V) data. More recently NaradaBrokering has also been used to provide archival and reliable delivery of messages. Real-time constraints for A/V conferencing applications can vary anytime between 30-100 ms, depending on the jitter in inter-packet arrivals in these streams. Packets in these streams generated at different locations can be buffered (either during replay or real-time), and time-ordered to provide an efficient collaboration session. If time-ordering among these streams is lost, it would be very unpleasant during the replay/real-time-play of these streams. The range that NTP provides is sufficient for such a collaboration environment.

The paper is organized in the following way: Section 2 describes related work. Section 3 introduces NaradaBrokering messaging system. Section 4 describes the design and implementation of time service provided to NaradaBrokering. Section 5 provides test results. And we will give conclusion and future work in section 6.

2. Related work

Different approaches exist to synchronize the events in a distributed system. One of them is to use logical clocks, which was first presented by Lamport [11], and the other one is to synchronize the system clocks so that clocks running on different machines are synchronized with each other.

Using logical clocks guarantee the order of events among themselves. They do not need to run at a constant rate, but they must increase monotonically. Using Lamport timestamps, Lamport synchronizes logical clocks by defining a relation called "happens-before". Vector clocks [12-13] have also been introduced because Lamport timestamps cannot capture causality. But a major drawback is that vector clocks add a vector timestamp, whose size is linear with the number of processes, onto each message in order to capture causality. Vector clocks thus do not scale well in large settings.

Various algorithms have been devised to synchronize physical clocks in a distributed environment. In Cristian's algorithm [14], all of the machines in the system synchronize their clocks with a time server. In this algorithm, each machine asks the current time to the time server by sending a message to it. The time server responds to that message including its current time as fast as it can. Time server in Cristian's algorithm is passive. In the Berkeley algorithm [15], time server is active. It polls every machine periodically, and then computes the average time based on the times received from them and tells them to adjust their time to the recent computed time. Another type of synchronization algorithm is the averaging algorithm, which are also known as decentralized algorithms. An example to a decentralized algorithm might be to average the time received from other machines. Each machine broadcasts its time and when the resynchronization interval is up for a machine it computes the average time from the samples received in that time interval. In averaging those times one might just take the average of them or can discard the *m* highest and *m* lowest samples and then average the rest.

Hardware approaches [16-19], are also available. But hardware approaches might require some custom made hardware components such as Network Time Interface (NTI) M-Module [19]. NTI M-Module is a custom VLSI chip that has interfaces to GPS receivers. It uses the time received by GPS receivers to achieve synchronization. Obviously hardware solutions are expensive and might require changes to the underlying platform.

---

[1] Stratum number is an integer indicating the distance from the reference clock. Stratum-0 is the reference clock.

There are also other solutions available, but one that we are most interested in is the Network Time Protocol (NTP). NTP is one of the most widely used algorithms in the Internet and can achieve to 1-30 ms accuracy. However, this accuracy also depends on the roundtrip delay between the machine and the time server supplying the time service. The difference between the delay from machine to time server and the delay from time server to machine also contributes to the accuracy of the offset computed. NTP achieves this accuracy by using filtering, selection and clustering, and combining algorithms to adjust the local time. NTP receives time from several time servers. Filtering algorithm selects the best from a window of samples obtained from a time server. Selection and clustering algorithms pick best *truechimers* and discard the *falsetickers*. Combining algorithm computes a weighted average of the time offset of the best truechimers. An adaptation of NTP, Simple Network Time Protocol (SNTP) [20] can also be used to synchronize computer clocks in the Internet. The major difference between SNTP and NTP is that SNTP does not implement the algorithms mentioned above. It just uses the time obtained from a time server.

NTP daemons are implemented for Linux, Solaris and Windows machines and are also available online, [21]. These NTP daemons sometimes adjust the system clock every 1 sec. This synchronization interval might be too frequent, and can place strains on bandwidth and CPU utilization. The decision on deciding the synchronization interval is also another issue. Setting this to a high value might cause the clocks getting out of sync too much while setting this to a low value might cause performance degrade. If two clocks need to be synchronized with δt time apart, then synchronization interval, Δt, should be chosen as $\delta t/(\sigma_1+\sigma_2)$ where σ_1 is the drift rate[2] of clock1 and σ_2 is the drift rate of clock2 [22].

3. NaradaBrokering

NaradaBrokering is an event brokering system designed to run on a large network of cooperating broker nodes. Communication within NaradaBrokering is asynchronous and the system can be used to support different interactions by encapsulating them in specialized events. NaradaBrokering guarantees delivery of events in the presence of failures and prolonged client disconnects, and ensures fast dissemination of events within the system. Events could be used to encapsulate information pertaining to transactions, data interchange, system conditions and finally the search, discovery and subsequent sharing of resources.

In NaradaBrokering, stable storages existing in parts of the system are responsible for introducing state into the events. The arrival of events at clients advances the state associated with the corresponding clients. The guaranteed delivery scheme within NaradaBrokering does not require every broker to have access to a stable store or DBMS. Stable stores can fail but they do need to recover within a finite amount of time. During these failures the clients that are affected are those that were being serviced by the failed storage. Currently in NaradaBrokering we have both SQL and file based implementations of the storage services needed by the robust delivery algorithms. NaradaBrokering is Java Message Service (JMS) compliant [23], supports P2P interactions [9], and audio-video conferencing [24-26], and communication through firewalls among others.

4. Time Service

Time service provided within NaradaBrokering (0.95) currently implements NTP version 3. NaradaBrokering is also an open source project which is implemented completely in Java. A configuration file, which contains time server addresses and other required NTP parameters, is provided to the time service. We impose no limit on the number of time servers

[2] The maximum drift rate of a hardware clock is provided by the manufacturer and indicates how many microseconds a hardware clock drifts apart from real-time per second.

that can be specified in the configuration file. In addition to these parameters, the interval that the time service should run is also specified in the configuration file. The value of this parameter affects the synchronization range of the computer clock. If it is too high, computer clock may be way of out of sync, and if it is too low it may utilize too much system and bandwidth resources.

It should be noted that time service does not change the system time. That is, unlike NTP daemons provided, it does not set system time to a new value. There are two reasons for this. First, in order to change the underlying system clock, one needs administrative privileges. This is not possible for clients without administrator privilege. The second reason is that the objective of this time service is to provide a mechanism to be able to time-order the events generated within NaradaBrokering without affecting the system and other applications running on the same machine. A call to the Time Service returns the adjusted time. It achieves this by keeping the offset in a separate variable. The `getTimestamp()` method returns the time obtained from the local system time adjusted with this offset in milliseconds. When time service starts it computes the first value of the offset. After this initialization, time service updates the offset at regular intervals based on the parameter specified in the configuration file. All events generated anywhere, by any entity, within the system utilize their Time Service to timestamp events.

4.1 Initialization

The initialization step is an important step in achieving synchronization. This initialization step is a blocking operation during the bootstrapping of NaradaBrokering services and should complete within a few seconds. The initialization step also uses NTP, but instead of using the interval specified in the configuration file, it waits 500 ms between its attempts, which is also limited for a total of 5 seconds.

4.2 NTP Implementation

We use NTP version 3 as the basis for our NTP implementation. We have chosen NTP instead of SNTP, because NTP implements advanced algorithms to filter the NTP message obtained from the time server and implements selection algorithms to those received NTP messages. A NTP message can be received from any number of NTP time servers. Our NTP implementation is as follows:

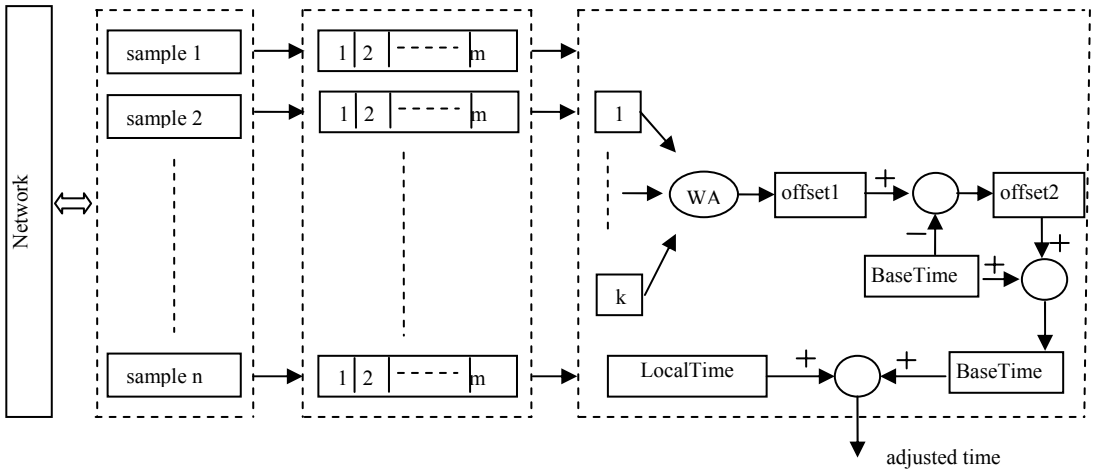
First step involves getting samples from NTP time servers. NTP client sends NTP messages to those servers specified in the configuration file one by one. That is, it sends a NTP message and waits for the response. This message is a datagram packet and the NTP message sent may be lost in the network. Because of this, NTP client sets a timeout to the UDP socket. The timeout chosen for our implementation is 500 ms. This step requires that NTP replies be received from at least half of the servers. Offset and roundtrip delay are calculated at this step. Upon receiving a NTP packet, a NtpInfo object is generated which contains NTP parameter values, i.e. offset, roundtrip delay, dispersion, timestamps, etc.

After collecting NTP samples from servers, the second step is to use the NTP filtering algorithm. The filtering algorithm checks timestamps to validate the received NTP message. It keeps a register dedicated for each time server and records the NTP samples received from that server. This register only keeps a specified amount of samples, and uses a First-In-First-Out (FIFO) scheme to accommodate new samples the register is full. The window size of this register is specified in the configuration file and has an effect on the computed offset.

After the previous steps are successfully completed, the new offset is computed using selection and combine algorithms as explained in the NTP specification. A clustering algorithm explained in NTP specification is then used to find a candidate list from which the new offset would be computed. Clustering algorithm uses stratum value obtained from NTP

message and synchronization distance computed from the NTP parameters of the related server to construct this candidate list. The result of clustering algorithm provides a candidate list, which contains the synchronization distance and the offset obtained from each time server. The combining algorithm then computes the weighted average of this candidate list according to the synchronization distance. So a server with a small synchronization distance has more impact on the new offset.

The steps explained in this section can be depicted as in figure 1. There are two offset values indicated in figure 1, offset1 and offset2. offset1 is the offset value computed with NTP. Since we do not change the system clock, we need to keep a variable named as BaseTime, which keeps the offset computed with NTP in an earlier computation. offset2 is the difference between the BaseTime and the offset computed by NTP. It can also be viewed as change of the offset.



offset1: Offset computed with regard to local computer time
offset2: Offset computed with regard to adjusted time

Figure 1: Steps taken in computing offset using NTP algorithms

4.3 Updating Offset

Unfortunately, calculating offset as in previous steps is not sufficient to achieve synchronization. A newly computed offset may not be used as is. The order of the messages generated at the local computer should also be preserved. Let's say message m1, which is the latest message before the new offset is calculated, has timestamp of t1 and message m2, which is the first message would use the new adjusted time, has a timestamp of t2. Then t2 cannot be less than t1. Because in that case, ordering algorithms may conclude that m2 was generated before m1, which is not correct. So the offset is not updated if the new value would cause such an inconsistency in the order of the messages. In this case we do not update the timestamp so long as the discrepancy persists. The pseudo code can be written as below;

```
long getTimestamp() {
        timestamp = LocalTime + BaseTime;
        if (timestamp >lastTimestamp)
                lastTimestamp = timestamp;
        return lastTimestamp;
}
```

5. Test Results

We have done tests on several linux machines. The test duration is 48 hours. The interval for updating offset is about 30 seconds. There are 8 time servers specified in the

configuration file and all of them are stratum-1 NIST time servers. In the test results we also show the first offset value, standard deviation, average offset change, minimum and maximum values.

The tests mentioned in this section are done on computers available to Community Grids Lab, Indiana University, researchers. Computer clocks are not modified or preset before the tests. Test cases are given below.

i) darya.ucs.indiana.edu

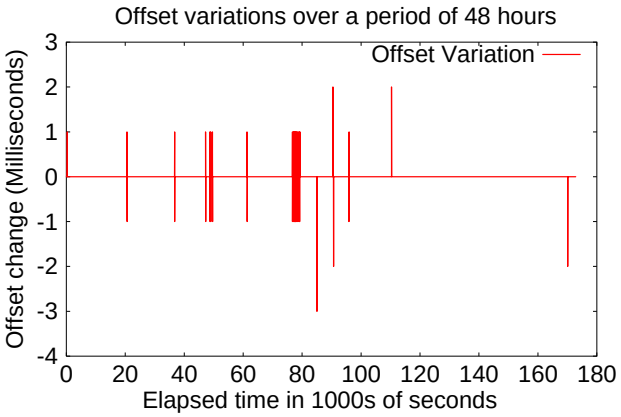


Figure 2: Change of offset with time for darya.ucs.indiana.edu

OS: Red Hat Linux release 7.3 (Valhalla)
CPU: AMD Athlon(tm)MP 1800+, 1533.42 MHz
Memory: 512 MB
JVM version: 1.4.1_03

initialization offset value	0 ms
standard deviation	0.11
average	-0.00018 ms
min value	-2 ms
max value	3 ms
total change	-1 ms
number of data	5690
total test duration	172800 sec

Table 1: Numeric values for darya.ucs.indiana.edu

ii) kamet.ucs.indiana.edu

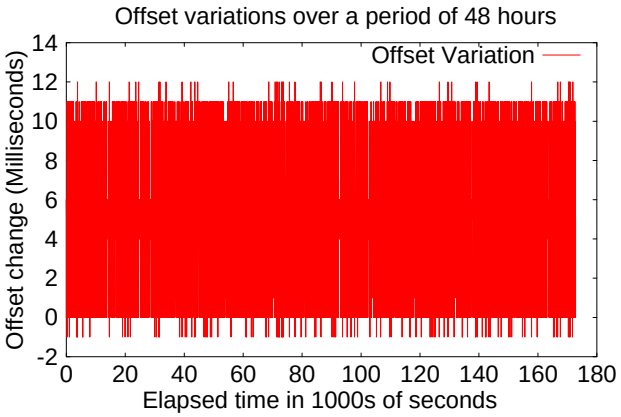


Figure 3: Change offset with time for kamet.ucs.indiana.edu

OS: Red Hat Linux release 9 (Shrike)
CPU: Intel(R) XEON(TM) CPU 1.80GHz
Memory: 1 GB
JVM version: 1.4.1_02

initialization offset value	1185869 ms
standard deviation	3.32
average	5.21 ms
min value	-1 ms
max value	12 ms
total change	29666 ms
number of data	5690
total test duration	172800 sec

Table 2: Numeric values for kamet.ucs.indiana.edu

iii) murray.ucs.indiana.edu

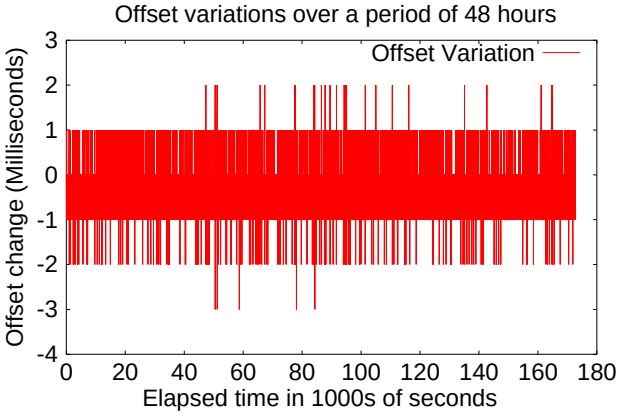


Figure 4: Change offset with time for murray.ucs.indiana.edu

OS: Red Hat Linux release 7.2 (Enigma)
CPU: Intel Intel(R) Pentium(R) III CPU family 1266MHz
Memory:1 GB
JVM version: 1.4.1-rc

initialization offset value	-139895 ms
standard deviation	0.71
average	-0.19 ms
min value	-3 ms
max value	2 ms
total change	-1060 ms
number of data	5690
total test duration	172800 sec

Table 3: Numeric values for murray.ucs.indiana.edu

5.1 Evaluation of Test Results

Over a period of 48 hours, totally 5690 times the offset is computed and the changes of offsets are shown in figures 2 – 4. First offset values, standard deviations, averages, minimum values , maximum values and total changes in the offsets for test cases i – iii are shown in tables 1 – 3 to give us some numerical ideas.

In test case (i), ntpd daemon is running. Among 5690 computed offset values only 24 of them are different than zero, which means that ntpd daemon running on the machine and our time service are very consistent with each other. The first value of the offset is also zero, because ntpd daemon is able to keep the machine synchronized. This ntpd daemon synchronizes its time with "time.nist.gov" time server.

In test case (ii), ntpd daemon is also running. But time server is set to "clock.redhat.com" for this server, which we tried to access using ping command and all requests are timed out. That is, it is not reachable. ntpd daemon has no effect on this machine. The change offset is between (-1) – (12) ms. The first offset value is 1185869 ms, which means that the clock is behind the real time by that many milliseconds.

In test case (iii), no ntpd daemon is running. The change of offsets in case (iii) is between (-3) - (2) ms. The first offset value for case (iii) is -139895 ms, which shows how much the clock in that machine is ahead of the real time.

If we pay attention to the average value of these offset changes, as indicated in tables 1 – 3, the machine in test case (ii) has a positive value and the machine in test case (iii) has a negative value. Also total changes for test case (ii) is positive, and for test case (iii) is negative. From this we can conclude that the clock running on machine for test case (ii) is a slow clock, because positive adjustment is done to the underlying system clock and the clocks running on machine for test case (iii) is a fast clock, because negative adjustment is done to the underlying system clock. Note that the adjustments needed for cases, (ii, iii), are different. This also shows that the clock rates on the machines are different. Since a ntpd daemon is running on the machine for test case (i), we avoid making such a conclusion regarding that clock.

5.2 Inter Client Discrepancy

We have also tested the discrepancy between two of the machines that are running the time service explained in this paper. The test environment is established as shown in figure 5. In order to receive requests from a remote client, we also implemented a NTP server. We also implemented a client that uses the NB Time Service time and is capable of sending NTP requests to the server.

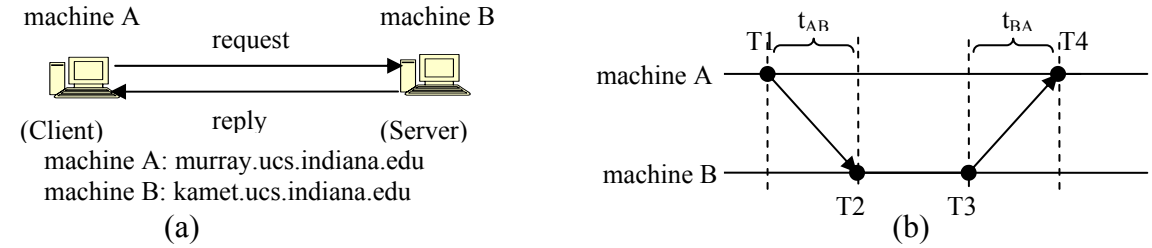


Figure 5: (a)Computers that run NTP server and NTP clients (b) Initiation and end of a time request between machine A and B.

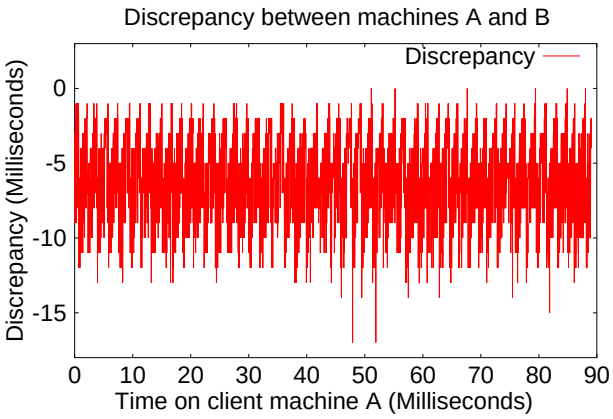


Figure 6: Discrepancy between machine A (murray.ucs.indiana.edu) and machine B (kamet.ucs.indiana.edu)

standard deviation	2.94
absolute average discrepancy	5.6 ms
absolute minimum discrepancy	0 ms
absolute maximum discrepancy	17 ms

Table 4: Numeric values for discrepancy between machine A and machine

As shown in figure 5b, a request originates from client and is received by server. Then server timestamps after it receives it and before it sends it back to the client. Client timestamps the reply. The delay involved in this process is the transmission time (t_{AB} and t_{BA}) and the process time on the server (T3-T2).

The discrepancy (ΔT) between these two machine clocks can be approximated as $\Delta T=0.5*(T2+T3-T1-T4)$, if t_{AB} and t_{BA} are ignored. Figure 6 shows this discrepancy computed. This approximated discrepancy is also found to be same with the offset computed by NTP. The discrepancy is measured every 30 seconds and test duration is about 25 hours.

6. Conclusion and Future Work

The first offset values in all of these computers are different and this demonstrates the need to synchronize these clocks. Hence one cannot rely on the underlying clock and use the system clock to timestamp the events generated in messaging systems. Each of these machines also have different changes in offset patterns, which shows that if these clocks are let running without periodic synchronization, they can be out of sync by large amount all the time.

We used NTP to achieve this synchronization level. NTP allow us to synchronize the machine clocks with atomic time servers available all over the world. Servers distributed in a wide geographic area would be synchronized with each other in a limited range. To achieve this, of course, time servers should use the closest time servers available to them. Since we know that those atomic time servers are tightly synchronized with each other, we would also be able to synchronize computer clocks.

Our future work is to implement a buffering service to NaradaBrokering messaging system, which will use this time service in order to sort to sort events/messages based on the timestamp values they carry.

7. References

[1] D. Deeths, G. Brunette. Using NTP to Control and Synchronize System Clocks - Part I: Introduction to NTP. Sun BluePrints™ OnLine - July 2001

[2] D.L. Mills. Network Time Protocol (Version 3). Specification, Implementation and Analysis. Internet RFC 1305. (March 1992)

[3] D.L. Mills. Internet time synchronization: the Network Time Protocol. IEEE Trans. Communications 39, 10 (October 1991), 1482-1493.

[4] National Institute of Standards and Technology (NIST), http://www.boulder.nist.gov/timefreq/index.html

[5] U.S. Naval Observatory (USNO), http://tycho.usno.navy.mil/ntp.html

[6] The NaradaBrokering Project at the Community Grids Lab: http://www.naradabrokering.org

[7] Shrideep Pallickara and Geoffrey Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003.
[8] Geoffrey Fox and Shrideep Pallickara. An Event Service to Support Grid Computational Environments Journal of Concurrency and Computation: Practice & Experience. Special Issue on Grid Computing Environments. Volume 14(13-15) pp 1097-1129.
[9] Geoffrey Fox, Shrideep Pallickara and Xi Rao. A Scaleable Event Infrastructure for Peer to Peer Grids. Proceedings of the ACM Java Grande ISCOPE Conference 2002. pp 66-75. Seattle, WA.
[10] Ahmet Uyar, Shrideep Pallickara and Geoffrey Fox. Towards an Architecture for Audio Video Conferencing in Distributed Brokering Systems. Proceedings of the 2003 International Conference on Communications in Computing.
[11] L. Lamport. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, vol. 21, no. 7 pp. 558-565, July 1978
[12] C.J. Fidge. Logical Time in Distributed Computing Systems. IEEE Computer. vol.24, no.8, pp.28–33, 1991.
[13] F. Mattern. Virtual Time and Global States of Distributed Systems. Proc. of Parallel and Distributed Algorithms, pp. 215–226, 1988.
[14] F. Cristian. Probabilistic clock synchronization. Distributed Computing. 3:146–158, 1989.
[15] R. Gusella, S. Zatti. The accuracy of clock synchronization achieved by TEMPO in Berkeley Unix 4.3BSD. In IEEE Transactions on Software Engineering. Vol. 15, pp. 847-853
[16] P.B. Danzig, and S. Melvin. High resolution timing with low resolution clocks and a microsecond timer for sun workstations. ACM Operating Systems Review 24, 1 (January 1988), 23-26.
[17] P. Ramanathan, D. D. Kandlur, and K. G. Shin, Hardware assisted software clock synchronization for homogeneous distributed systems. IEEE Trans. Computers. vol. 39, no. 4, pp. 514-524, April 1990
[18] P. H. Dana. Global positioning system (gps) time dissemination for real-time applications. Real-Time Systems Journal, 12(1), 1997.
[19] M.Horauer, U.Schmid and K.Schossmaier. NTI: A Network Time Interface M-Module for High-Accuracy Clock Synchronization. Proceedings of the 6th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS). Orlando Florida USA, March 30 - April 3 1998.
[20] D.L. Mills. Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI. Internet RFC 2030, October 1996
[21] http://www.ntp.org/
[22] A.S. Tanenbaum, M.van Steen. Distributed Systems, Principles and Paradigms. Textbook, Prentice Hall, 2002
[23] Mark Happner, Rich Burridge and Rahul Sharma. Java Message Service Specification. Sun Microsystems. 2000. http://java.sun.com/products/jms.
[24] Wenjun Wu, Geoffrey Fox, Hasan Bulut, Ahmet Uyar, Harun Altay. Design and Implementation of a Collaboration Web-services System. to be published in special issue on Grid computing in Journal of Neural Parallel and Scientific Computations (NPSC)
[25] Wenjun Wu, Hasan Bulut, Ahmet Uyar, Geoffrey C. Fox. A Web-Services Based Conference Control Framework for Heterogenous A/V Collaboration. 7th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA 2003) August 13-15, 2003 Honolulu, Hawaii, USA
[26] Geoffrey Fox, Wenjun Wu, Ahmet Uyar, Hasan Bulut, Shrideep Pallickara Global Multimedia Collaboration System. Proceedings of the 1st International Workshop on Middleware for Grid Computing co-located with Middleware 2003 on Tuesday, June 17, 2003 Rio de Janeiro, Brazil