International Journal of Computational Intelligence Research.
ISSN 0973-1873 Vol.4, No.1 (2008), pp. 1–8

Obtaining Performance Measures through Microbenchmarking in a Peer-to-Peer Overlay Computer*

Paolo Bertasi, Mauro Bianco†, Andrea Pietracaprina, and Geppino Pucci

Dept. of Information Engineering,
University of Padova, Padova, Italy
{bertasi,bianco1,capri,geppo}@dei.unipd.it

Abstract: The availability of enormous amounts of un-used computing power and data storage over the In-ternet makes the development of a globally distributed computing platform, called Overlay Computer (OC), at-tractive for the industry, the scientific community, and also the general user. In order to be effective, an OC must be equipped with tools for estimating and optimiz-ing the performance of applications. The tools must rely upon performance metrics of the key factors impacting performance of a distributed application, such as inter-connection *latency* and *bandwidth*, and available *comput-ing power* at the nodes. This paper first reviews the state-of-the-art on performance benchmarking approaches in globally distributed systems, including performance-conscious P2P systems. Later, it addresses the problem of developing a suite of microbenchmarking experiments aimed at providing the basic functionalities of a measurement tool for a P2P-based OC platform. Results show that such a measuring system should take into account the communication patterns generated by the applications in order to provide useful performance insights.

1. Introduction

The impressive amount of computationqal resources potentially available through the Internet has stimulated several attempts at making them profitable for vast scale applications, the first and most popular being Peer-to-Peer (P2P) file-sharing applications. The opportunity to exploit idle cycles of connected computers has also led to the development of several world-distributed applications, such as SETI@Home [SETI], GIMPS [GIMPS], among the others. While these applications are essentially based upon a simple producer-consumer paradigm, they are a witness of the potential offered by wide-area network distributed computing also for other, more general applications [LZ+04]. The parallel computing platform that would actually deploy the computing and storage resources needed for this purpose, and whose network topology is embedded within the Internet is generally referred to as an *Overlay Computer* (OC).

To be successful, an OC must provide higher performance and capabilities than the individual computing equipment available to the user. To this purpose, it is necessary to provide the user with tools for estimating the effectiveness of design choices on such a distributed computer. The tools must rely upon a measuring tool aimed at providing a reasonably accurate estimate of the key factors impacting performance of a distributed application, such as interconnection *latency* and *bandwidth*, and available *computing power*.

The approach pursued in this paper is the use of *microbenchmarking techniques* to measure basic performance characteristics of a P2P-based OC. In turn, these techniques can be employed for improving the topology of the OC embedding onto the Internet, which has the potential of yielding high pay-offs in terms of performance. Furthermore, it may be useful to have a system which provides a quantitative assessment of specific OC capabilities, e.g., the ability to efficiently route certain communications patterns of widespread use. Finally, performance measurements may also aid *applicationside optimizations*, e.g., the efficient spawning of a given application over a suitable set of OC peers.

This work was supported in part by MIUR of Italy under project MAINSTREAM and by the EC/IST Project 15964 AEOLUS. A preliminary version of this work was published in the Proceedings of the Proceedings of the First International Conference on Complex, Intelligent and Software Intensive Systems, Vien, April 2007.
†Mauro Bianco is currently affiliated at Computer Science Dept., Texas A&M University, College Station, TX, USA.

This paper offers two main contributions. The first is to provide an overview on performance measurement techniques for related parallel or distributed platforms such as clusters, the Internet, and P2P systems. The second is to provide a preliminary set of experimental results to serve as *proof-of-concept* for the development of a more extensive microbenchmarking tool for an OC. The proposed tests aim at identifying both measurement techniques and key performance factors that any such tool will have to embody.

The rest of the paper is structured as follows: Section 2 presents the above-mentioned overview. Section 3 presents the tests implemented as a preliminary effort towards a more complete microbenchmarking tool, together with results of performance measurements executed on a local area network running JXTA. Finally, Section 4 draws some conclusions and describes future work.

2. Techniques for Performance Measurement and Improvement: an Overview

This section is dedicated to an overview of techniques used in related systems to measure and/or improve performance in contexts closely related to overlay computing, such as clusters of workstations, the Internet, and P2P systems.

Overview of performance measurements techniques. Several efforts have been carried out to provide accurate performance measurement tools for parallel platforms using MPI. The mpptest utility, developed in [GL99], consists of a suite of programs that provide reproducible performance measurements relative to low-level MPI primitives as well as to more complex communication patterns. In [GL99] the authors also point out the perils of badly designed tests, the most relevant being 1) *confusing total bandwidth with point-to-point bandwidth* 2) *using communication patterns different from those featured by the application*, 3) *measuring under a single communication pattern*, and 4) *timing events that are short relative to clock resolution*. In our experiments, we have been careful in avoiding these pitfalls.

The problem of obtaining accurate measurements without incurring in long testing sessions is addressed in [GC01] where the *MPIBench* tool is developed to measure performance of MPI-based computations. Although reducing testing time is also a highly desirable objective for an OC, the approach taken in [GC01] relies on the use of high precision timers which, however, would require substantial effort to be implemented in a OC. In fact, the tradeoff between accuracy and measurement overhead is a major challenge for an OC.

Another platform for parallel and distributed computing is the Grid. Scheduling decisions on the Grid are based essentially on performance declarations of the (trusted) institutions participating in Grid initiatives. In [TD05] the *Grid-Bench* suite is described which addresses the problem of measuring performance to alleviate the management work of institutions participating in a Grid. More interestingly to our purposes, in this paper the minimization of testing time is stressed, along with the reliability of the obtained results. The measurement system also manage the access to performance information by users. Organizing the performance information of an OC can be more cumbersome, since the users cannot rely on stable institutions, but still potentially useful to reduce overheads.

The availability of high bandwidth over the Internet has made possible the development of several kinds of distributed services, many of them relying on performance metrics to achieve good performance. This has inspired numerous research projects aimed at getting performance parameters out of the Internet, with the double objective of minimizing the testing time while avoiding to overload the network itself. Typically, measures of interest for the Internet are latency and bandwidth, where latency is often referred to as *distance* and depends on the topology of the network at a given instant.

The simplest way to measure the latency is through *ping time*, which, however, may require many repeated experiments, since ping times are subject to a high variance. Since distances depend only on topology, many works try to characterize the Internet using fictitious coordinate systems whose aim is to allow the estimation of ping time without prior communication, thus avoiding time-consuming probing: what a node need to know are the coordinates of the remote node to compute the distance of that node. A coordinate system must provide an accurate embedding of the internet hosts in a low-dimensional space with a distance metric, with additional requirements for scalability and dynamic adaptation. Coordinates can be evaluated by measuring distances from some special hosts called *landmarks*, [NZ02, KZ04], or in a totally distributed fashion, as in [RD01, DL$^+$04]. A well known coordinate system for the Internet is *Vivaldi* [DC$^+$04]. Vivaldi treats the internet as a self-adjusting system of springs, where distances are dynamically updated in a totally distributed fashion, thus avoiding the use of static landmarks that pose scalability and fault-tolerance problems. Vivaldi is also employed in some actual P2P implementations to improve the quality of the overlay network [RD01, DL$^+$04].

Measuring communication bandwidth appears more difficult than estimating latency, since the bandwidth available between two hosts depends on several (global) aspects, such as the quality of the path connecting the two nodes, the amount of competing traffic along that path, the load at the end-points, etc. Since the load profile of a wide-area network tends to exhibit a slowly-changing profile, *instantaneous* measures of bandwidth can still produce results which are valid for a certain time interval. When measuring bandwidth, one is faced with the problem of defining the *type of bandwidth* that is going to be measured (recall the first pitfall mentioned by [GL99]). In fact, point-to-point unidirectional bandwidth, bidirectional, cumulative, or bisection bandwidth, are all measures that relate that describe different capabilities of the interconnection, hence it is often difficult to compare the quantitative results obtained for different measures. We address this problem for our OC infrastructure in Section 3.

Many works in the literature measure bandwidth in a wide-area network by defining the *bottleneck link bandwidth*, that is, the minimal bandwidth available along the links in a path between two hosts. This measure is ideally related to the hardware parameters of the links in a path, so its measure is not dependent on traffic conditions during the

measure. In [HS05] an efficient strategy is proposed to obtain the $O(N^2)$bottleneck link bandwidths on an N-node distributed system, which exhibits an overhead linear in the number of nodes and requires only limited cooperation among them. The resulting measures are very descriptive, since they provide an estimate of the maximum possible bandwidth between any pair of nodes. However, the infrastructure to be provided is quite complex, and some of the hypotheses upon which the whole approach relies may be difficult to satisfy in an OC context.

Topology awareness techniques for performance improvement on P2P systems. P2P systems employ several techniques to provide better performance by exploiting topological characteristics of the underlying network. Recent works have pursued a quantitative approach to measure the gains produced by *Topology-awareness*. The work in [DL+04] reports on an extensive experimentation of *DHash++*, which implements a Distributed Hash Table (DHT) based on *Chord* [SM+01]. The authors experiment with several different lookup algorithms and evaluate the resulting performance in terms of lookup time and throughput. Both measures result to be highly affected by locality-awareness of the protocols, which can halve the average time of a request. We remark that only latency is taken into account as a measure of distance, since the data to be retrieved is as small as 8*K*Bytes in the application under examination, and this makes the throughput be highly dependent on latency more than on bandwidth.

Earlier works also tried to characterize P2P file sharing application workloads in order to evaluate the improvements that topology-awareness would provide if included in the implementation. One such work is [GD+03], which analyzes the traces of Kazaa (uncached) traffic within the network of the University of Washington. The results suggest that, if content were cached, thus providing locality awareness, then about 60% of external bandwidth would be saved.

Trace analysis is also employed in [LP03], in the context of the *PeerMetric* project. The paper suggests that lasthop bandwidth is a major bottleneck and that latency-based optimization of the overlay embedding may be somewhat a less impacting issue, especially for bandwidth intensive P2P applications.

One of the first implementations of a topology-aware P2P system is *Pastry* [RD01]. In Pastry, a node comes with a random ID that identifies the position of the node in the overlay topology. To allow for locality optimization, a node can choose the nearest among *k*potential neighbors (nodes whose ID is close to the node's ID). The proximity metric chosen by Pastry is the number of Internet hops (an approximation of latency) in the path between two nodes.

The work presented in [RH+02] proposes a strategy to insert topology-awareness into CAN [RF+01] (the idea is however somewhat more general). In CAN the ID space is represented by a *d*-dimensional unitary cube in the ordinary Euclidean space. Upon joining, a node is given an ID within a large region of the cube that is then partitioned accordingly. In the method proposed in [RH+02], the node set is partitioned into *bins*, and a new node wanting to join must find a suitable bin where to fall. Then the node probes a set of predefined unconscious landmarks (a web server, a DNS server, etc.) in order to derive a *bin identifier*, that is, the list of the landmarks sorted by increasing distance. By looking for other nodes with the same bin identifier, performance can be improved by placing the node in the larger region among those of the nodes within its bin. The presented results show that topology-awareness can significantly help in improving performance (measured in terms of latency stretch).

GIA [CR+03] implements a strategy to make *unstructured* P2P systems topology-aware. Unlike *structured* P2P systems, unstructured ones allow users to retrieve objects that match partial queries. Gnutella is an example of this approach where an incoming query in a peer is matched against local items and forwarded to other peers in the overlay network. Gnutella, however, does not optimize for locality, hence a neighbor in the overlay network can be arbitrarily distant in the physical network. GIA adapts itself to the underlying network and to the peer's *capacity*, i.e., the number of queries that a peer can process without being overloaded. Lookup protocols are then designed to guarantee an even utilization of the capacities of the peers by employing a token-based *search protocol* that relies on a *biased random walk*, where the next node in the path is chosen among the highest capacity neighbors for which the peer has a token. Results indicate that topology adaptation improves performance by orders of magnitudes with respect to the basic Gnutella system. From the point of view of this report, it is important to note that this is one of the few papers that also take into account the computational power of the peer, along with the traditional latency and bandwidth parameters.

The idea to organize the network with emphasis on high-capacity nodes is also adopted in [SGL04]. The paper describes a method to build a hierarchical overlay topology where nodes are classified again in terms of their capacity. The nodes with highest capacity form a *backbone* structure, while lower capacity nodes refer hierarchically to higher capacity ones. The resulting topology is tree-like, reminiscent of a *fattree* with additional edges mainly for fault-tolerance. Results indicate that substantial bandwidth savings can be attained with respect to a random topology. Capacity is intentionally left undefined in [SGL04], thus allowing developers to adapt the definition to the application needs.

Another aspect impacting the overall performance of a P2P system is *churn*, that is, the rate at which peers join and leave the system, since maintenance overhead increases with churn. The work in [1] includes a description of Bamboo, a DHT that explicitly addresses the problem of routing performance under heavy churn. Results indicate that topology awareness attains lower latency under high churn, thus proving to be beneficial also in these conditions.

Few works try to face the problem of topology-awareness

for P2P computing (rather than file-sharing) systems, since there are not many such platforms yet. *Zorilla* [DNB06] is a prototype P2P supercomputing platform. An algorithm similar to the one implemented in Gnutella is used by Zorilla for discovering peers when allocating jobs. The scheduling algorithm automatically sets a radius for constraining the search of peers in the neighborhood of the initiating peer. Locality awareness is guaranteed by the topology-awareness of the overlay topology which is obtained using Bamboo. Zorilla's peers that are close in the overlay topology are also close in the physical topology.

Zorilla tries to keep the spawned nodes close to the node that initiates the job submission, rather than randomly distributing them, as they are in Gnutella. This feature speeds up the initial phase of moving input files from the submitting node to the workers and the final collection of the results. While Zorilla's approach may be reasonable, for special applications (e.g., the case of very long computations not featuring heavy I/O), it may be more convenient to confine the computation into a subnetwork which does not contain the initiator, since the initiator itself could be in a sub-network that is not able to execute efficiently the pattern of communications of the particular application.

Table 1 : Computers used to perform experiments classified as *slow* or *fast*.

Cat.	CPU	Clock Speed	O.S.	Distribution
Slow	Pentium III	800 MHz	GNU/Linux 2.6.18	Debian Etch
Slow	Pentium III	866 MHz	GNU/Linux 2.6.18	Debian Etch
Slow	Pentium III	866 MHz	GNU/Linux 2.6.18	Debian Etch
Fast	Pentium 4 HT	2.4 GHz	GNU/Linux 2.4.20 smp	Red Hat 9
Fast	Pentium 4	1.7 GHz	GNU/Linux 2.6.18	Debian Etch
Fast	Pentium 4 HT	3 GHz	GNU/Linux 2.6.18 smp	Debian Etch

3. Design and implementation of initial tests

This section describes a preliminary suite of experiments aimed at identifying the main characteristics that impact performance in a P2P system based on JXTA [Gra02], a P2P API built over JAVA which is becoming popular as a *de facto* standard for P2P platforms development. The tests have been implemented using the *JXTASocket* interface, which provides reliable bidirectional communications. Lower level interfaces (e.g., *Pipes*), although faster, have not been used because of their unreliability. Our objective is to measure key performance quantities at the user level, such as *latency*, *bandwidth*, and *computing power*. The machines employed during the experiments are identified as *fast* (>1.5GHz processors) or *slow* (<1GHz processors), according to the classification in Table 1. This distinction is necessary to evaluate the impact of the software layers on performance.

To measure latency, the core of the experiment is a simple *pingtest*, where a peer sends a small packet (8 bytes) to a selected peer, which then replies as soon as it receives the message. To filter out noise this process is iterated several times. The initiating peer then computes the round-trip time by averaging over the iterations. Since JXTA is based on Java, it is important to quantify the software overhead. For this reason we have measured ping times between fast and slow computers and also compared them against the times obtained using the ICMP protocol. In Table 2 the latency measurements are showed. The table reports two ping times: the first is the time for JXTA-level ping messages, while the second time is obtained by running the ping command between pairs of machines. It possible to note that JXTA-level ping is up to three orders of magnitude slower than the ICMP one, because of the software overhead introduced by JXTA, hence the fastest times are obviously those among fast computers. It is also possible to note the following asymmetry: JXTA ping time from a slow computer to a fast computer is lower than the one from a fast computer to a slow computer. A similar phenomenon is also visible in the unidirectional bandwidth measurement presented in Figure 1.

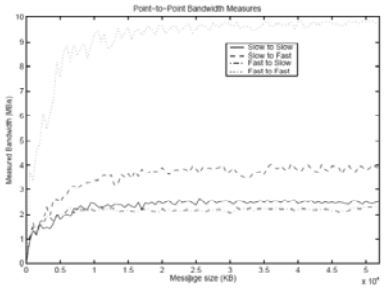


Figure 1 : Point-to-point bandwidth for different pairs of hosts.

Bandwidth is measured through several tests. Each peer involved in the benchmark measures the time for receiving and/or sending the amount of data assigned to it. Measuring communication times requires some attention. Since receives are posted before the actual data arrives, receiving time is measured from the reception of the first bytes till the end of communication, which are clearly identified instants. Since the IP protocol stack in fact does not allow the buffering of large amounts of data before actual transmission, a send can be considered to be *blocking*. Hence, sending times are measured by timing the beginning and the end of the application-level send operations. Communication time is the time a peer spends executing sends and/or receives. Bandwidth is then computed as the sum of the outgoing and incoming bytes divided by the measured communication time. This bandwidth measure

captures both the capability of the peer, the state of congestion of the network, and provides a uniform measure of bandwidth and a clear way to compare results of different communication patterns.

Table 2 : Application level vs ICMP ping times

Sender	Receiver	Appl. Lev. (ms)	ICMP (ms)
Fast	Fast	17.2	0.24
Fast	Slow	70.4	0.16
Slow	Fast	57.6	0.14
Slow	Slow	85.5	0.22

When measuring bandwidth, a simple clock alignment algorithm is employed to ensure that peers start the measurement roughly at the same time to stress the network. To perform clock alignment, a selected peer measures latencies from itself to all other peers involved in the test, and then sends them the time they have to wait before starting the experiment. Although quite naive, this simple algorithm has been deemed sufficient in our preliminary experiments to provide a lightweight synchronization of the peers.

The first measurement concerns point-to-point bandwidth and is implemented by letting one peer send a given amount of data to another peer and wait for an acknowledgement from the receiver. The measured time is the time to send all the data at sender side, and the time between the first byte is received and the receive finish at the receiver side. Several message sizes have been employed to identify the point where the bandwidth saturates. Results are depicted in Figure 1, that shows a high dependency on the underlying architecture. When employing fast computers, the bandwidth is quite close to the peak bandwidth of the LAN (i.e., 12.5MB/s) while slow computers dramatically hamper communication speed. We note that fast-to-slow communication is consistently worse than slow-to-fast communication, and indeed the former is even worse than slow-to-slow communication. To gain a deeper insight into this phenomenon, we ran a microbenchmark to send raw TCP data from one host to another. It turns out that the differences in unidirectional bandwidth for fast-to-slow and slow-to-fast are in this case negligible and explainable in terms of measurement noise. This confirms that once again the observed asymmetry is imputable to the JXTA software layer, where certain pairings between processors of different computational power induce a definite performance penalty.

We also measure the execution times of gather (i.e., all-to-one) and scatter (i.e., one-to-all) communication patterns, to evaluate the OC communication capabilities with respect to typical patterns arising in distributed applications. To measure the execution time of the gather pattern, the alignment algorithm is used to make all peers send the data to the *collector* roughly at the same time. The collector uses a number of receiving threads equal to the number of peers sending data to it. Similar considerations are valid for scatter: a *distributor* sends data to a number of involved peers. The distributor employs a number of threads equal to the number of receiving peers. All the bandwidth measurements have been carried out with different configurations involving fast and slow machines as collectors/distributors to provide insights over the software overheads. Since for gather and scatter a peer is exclusively a sender or a receiver, the measured times are the the times for sending and receiving the data, respectively.

Results for scatter are shown in Figures 2, as the size of the sent messages vary. Figure 2.a shows the results of a scatter performed from a slow sender to the others, while Figure 2.b depicts the scatter from a fast sender. Both the graphs exhibit the same trend. The difference in the graphs between the bars of the sender and the receivers is justified by the fact that the sender serializes the accesses to the network link. On the other hand, the bandwidths for scattering 1MB of data are all comparable. This is due to the fact that the packet size is too short to create congestion in the network. Hence, on our switched testbed platform, any direct measure of bandwidth must require at least 10MB of data. The same conclusion can be reached by looking at the unidirectional bandwidth experiments, noting that the bandwidth saturates when the messages reach a size of 10MB.

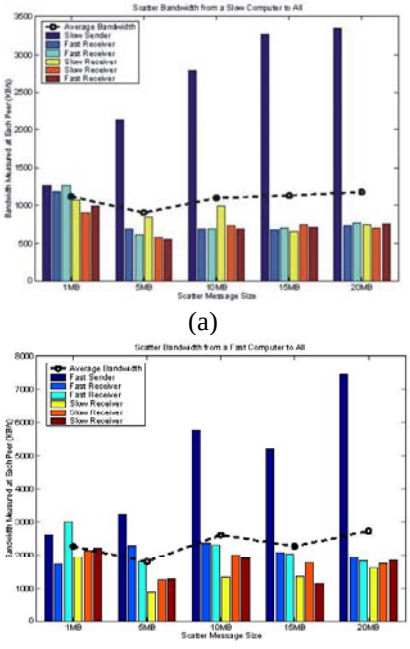


Figure 2 : Scatter measures from a slow sender to the others (Figure (a)), and from a fast sender to the others (Figure (b)). The plots show measured bandwidth at each peer along with the average value, as the size of the message received by each peer varies.

Clearly these observations are only valid in the context of our toy experiments, where the underlying interconnection is a homogeneous LAN. Different thresholds (but likely

similar behaviors) are to be expected when performing the experiments on a wide-area network in presence of a bottleneck link. We characterize this phenomenon in the following when the congestion is taken into account. Gather measurements involve similar considerations as scatter, we report Figure 3 for completeness, which shows the bandwidth measures while executing a gather to a fast peer. What can be noted here is that the receiver receives almost at the peak speed of the link, while the senders share the bandwidth of the link to the receiver.

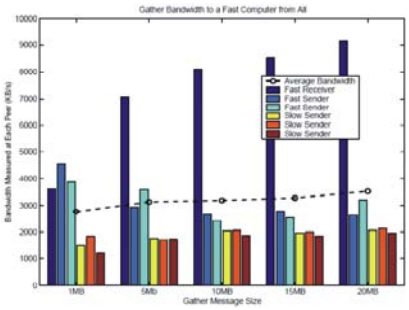


Figure 3 : Gather from to fast receiver from all the others varying message size.

A global measure of bandwidth can be provided by running an all-to-all communication pattern, where every peer sends the same amount of data to all the other peers. We remark that applications heavily employing this pattern may prove less attractive for execution on an OC, since, most likely, the availability of large network bandwidth would be required for the OC execution to be competitive. Figure 4 shows the result for an all-to-all pattern among 5 peers. Each peer has four receiving threads and four sending threads. Each peer measures the time spent in receiving and sending. Each peer computes the bandwidth by dividing the amount of data sent and received during the execution. The switched environment allows for a good use of the available bandwidth with respect to the gather and scatter communication patterns. As we show below, the presence of a bottleneck link highly affects the performance.

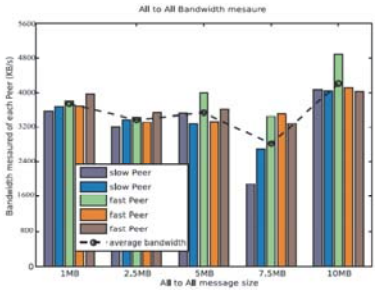


Figure 4 : All-to-All communication pattern among 5 peers.

Since the results shown above has been obtained in a 100Mb/s switched network, what has been measured is basically the software overhead of JXTA. To measure the effects of network congestion, another set of similar experiments had been carried out introducing an artificial band-width bottleneck as a 10Mb/s ethernet hub. We measured the bandwidth as before but placing a peer behind the hub, thus evaluating the impact of a single weak connection in the network, as depicted in Figure 5.

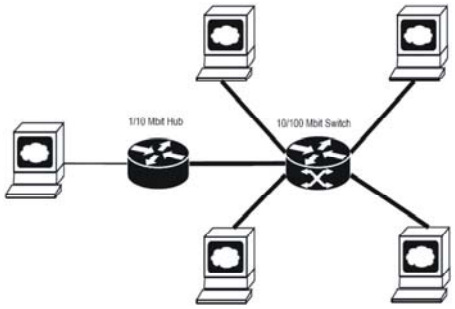
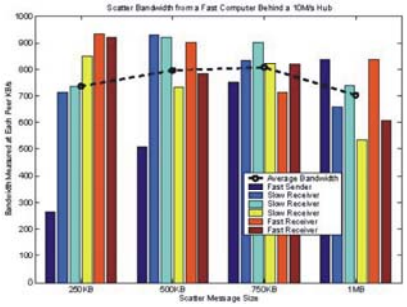


Figure 5 : Network configuration used for testing the presence of a bottleneck link. The bottleneck link has 1/10 the bandwidth of other links.

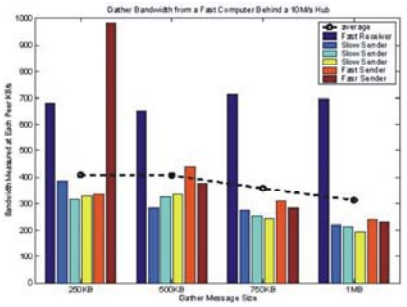
Figure 6.a shows the bandwidths measured at each peer while placing a fast peer behind the hub. In this example the sender has as many sending threads as the number of receivers. Each thread sends data independently from the other, but the sequential access to the network guarantees that a large portion of the peak bandwidth of the hub (1.25MB/s) is used. On the other hand, Figure 6.b shows the same setting while executing a gather communication. The receiver is behind the hub and has as many receiving threads as the number of senders. In this case, however, the senders try to access the bottleneck link concurrently, thus competing for reaching the destination. This degrades the overall performance with respect to a scatter communication pattern since, in the other case the competition were not present. Note that one of the fast senders consistently measures higher bandwidth for short messages. Awakening a process in this machine is faster that on the other, with respect to the latencies on the network. This let this peer acquire the network quickly and access the network in absence of contention. For shorter messages this leads to a greater measured bandwidth, while for long messages the randomization of the network access makes the peers to behave evenly. This phenomenon highlights the importance of the algorithm to synchronize the starting of the experiment. Indeed, it is caused by the alignment algorithm that does not take into account the runtime system overhead for awakening processes, which may have some impact in our setting where latencies are very small, and messages too short for the gather experiment.

In these experiments the message sizes had to be reduced with respect to the switched network. This has been possible since the lower bandwidth is saturated by far smaller messages, as the pictures highlight. As before the experiments has been repeated several times to filter out

noise. The results when a slow peer is placed behind the hub exhibits the same characteristics as when a fast peer is placed behind the hub. For completeness we include Figure 7, that shows the bandwidths while executing a scatter pattern from a slow peer behind the hub. This image is almost identical to Figure 6.a confirming that, when the bandwidth is the bottleneck the computational overhead becomes negligible.



(a)



(b)

Figure 6 : Measures of scatter from a fast sender to the others (Figure (a)), and gather to a fast receiver from the others (Figure (b)). The sender for the scatter and the receiver for the gather was connected to a 10MB/s hub.

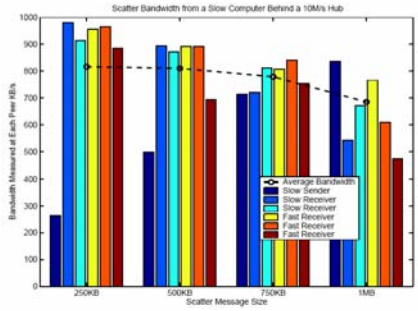


Figure 7 : Scatter from a slow sender behind a 10Mb/s hub varying message size.

When executing an all-to-all in presence of a peer behind the hub, what can be noted is that the performance is dominated by the bottleneck link. Figure 8 shows the bandwidths measured while performing a all-to-all among 5 peers. Differently from Figure 4, which shows a all-to-all in a switched network, in this experiment a slow peer has been placed behind a 10MB/s hub. The algorithm executed and the measured quantities are the same as in the fully switched network. What can be noted is that the bottleneck link makes the all measured bandwidths to be below the peak bandwidth of such a link. When placing a fast computer behind the hub the situation is not different, since the bottleneck link forces to slowdown the communication.

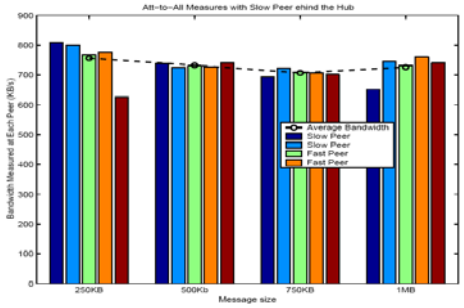


Figure 8 : All-to-All communication pattern among 5 peers. A slow peer is placed behind the hub. As it may be seen, bandwidth is dominated by the slow link.

To measure the computing power of a remote host in an uncooperative P2P setting we use a quiz-like approach [LZ+04]. The remote peer is required to perform a given computation and the execution time is measured by the inquiring peer (that needs to filter out communication time). In order to be effective, the computation needs to exhibit the following characteristics: 1) requiring both a small input and a small output; and 2) requiring a given amount of computation that cannot be avoided. The first requirement is necessary since we do not want the network to become a bottleneck, the second allows for a trusted measure, since the inquired peer cannot employ faster algorithms to provide the right answer. A computation that matches these requirements is given by a random number generator where the input is the seed and the output is the number generated after either a given number of iterations or a given amount of time. A fairly simple generator has been chosen, whose main iteration is seed=MOD(8121*seed+28411, 134456).

Figure 9 shows the results of CPU performance measurements. The plot shows the outcome of four tests: two of them fix the number of iterations to be performed, respectively on a slow and fast CPU; while the other two fix the computing time, respectively on a slow and fast CPU (in this latter case data are again plotted against the number of iterations performed in the allotted time). Even though the measurement is influenced by the fluctuations of the computational load of the inquired peer, the results indicate that such a test may be employed if a sufficiently large number of iterations are executed by the remote peer to deal with the clock resolution. For instance, from the plot we can say that 4 million iterations are sufficient to get a reasonable measure of computing power, which is equivalent to about 300ms of fast CPU time and 800ms of slow CPU time.

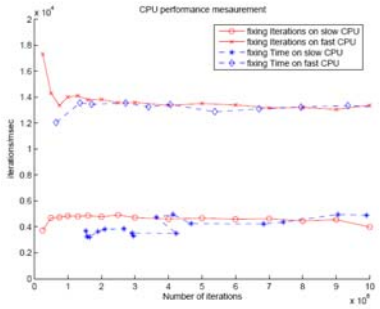


Figure 9 : CPU power measurements obtained fixing by the number of iterations and the total execution time on a slow and fast CPU, respectively.

4. Conclusions and future work

In this paper we dealt with the issue of measuring performance in an P2P-based OC, focusing on parameters such as the computing power of participating hosts, latency, and bandwidth for communication patterns arising in typical applications. After reviewing some relevant state-of-the-art measurement approaches employed in globally distributed system, we developed a suite of microbenchmarking experiments for measuring performance an OC built over JXTA. Preliminary results have shown that different patterns exhibit highly different behaviors, suggesting that the efficient execution of an application requires a careful choice of the executing nodes. Measuring systems should also provide adeguate countermeasures against selfishness and free riders, especially for what concerns estimating the computing power of a given node.

Future work will aim at extending the microbenchmarking suite to produce a complete measurement toolkit to be employed in an OC and to extend the experiments to large heterogeneous testbeds (such as PlanetLab) to fully assess the effectiveness of the proposed approach.

Bibliography

[CR+03] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker. Making Gnutella-like P2P Systems Scalable. In *Proc. SIGCOMM*, 407–418, 2003.

[DC+04] F. Dabek, R. Cox, F. Kaashoek, R. Morris. Vivaldi: A decentralized Network Coordinate System. *Proc. SIGCOMM*, 15–26, 2004.

[DL+04] F. Dabek, J. Li, E. Sit, J. Robertson, F. Kaashoek, R. Morris. Designing a DHT for Low Latency and High Throughput. Proc. of the 1st Symposium on Networked System Design and Implementation (NSDI '04), 2004.

[DNB06] N. Drost, R. Nieuwpoort, H. E. Bal. Simple Locality-Aware Coallocation in Peer-to-Peer Supercomputing. In *Proc. IEEE CCGRID*, 14, 2006

[Gra02] J.D. Gradecki *Mastering JXTA: Building Java Peer-to-Peer Applications*. Wiley, 2002.

[GIMPS] GIMPS: Great Internet Mersenne Prime Search. www.mersenne.org

[GL99] W. Gropp, E.L. Lusk. Reproducible Measurements of MPI Performance Characteristics. In *Proc. 6th European PVM/MPI Users' Group Meeting*, LNCS 1697, 11–18, 1999

[GC01] D. Grove, P. Coddington. Precise MPI performance measurement using MPIBench. In *Proc. HPC Asia*, 2001.

[GD+03] K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, J. Zahorjan. Measurements, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *Proc. 18th ACM SOPS*, 314–329, 2003.

[HS05] N. Hu, P. Steenkiste. Exploiting Internet Route Sharing for Large Scale Available Bandwidth Estimation. In *Proc. of Internet Measurement Conf.*, 187– 192, 2005

[KZ04] M. Kleis, X. Zhou. A Placement Scheme for Peer-to-Peer Networks Based on Principles from Geometry. In *Proc. 4th IEEE Intl. Conf. on P2P Computing*, 134–141, Aug. 2004.

[LP03] K. Lakshminarayanan, V.N. Padmanabhan. Network Performance of Broadband Hosts. Microsoft Research, Tech. Rep. MSR-TR-2003-15, 2003.

[LZ+04] V. Lo, D. Zhou, D. Zappala, Y. Liu, and S. Zhao. *Cluster Computingon the Fly*: P2P Scheduling of Idle Cycles in the Internet. In *Proc. 3rd Int.l Workshop on P2P Systems*, 227–236, 2004.

[NZ02] E. Ng, H. Zhang. Predicting Interne Network Distance with Coordinates-Based Approach. In *Proc. IEEE INFOCOM*, 170–179, 2002.

[RF+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. A Scalable Content-Addressable Network. In *Proc. ACM SIGCOMM*, 161–172, 2001.

[RH+02] S. Ratnasamy, M. Handley, R. Karp, S. Shenker. Topologically Aware Overlay Construction and Server Selection. In *Proc. IEEE INFOCOM*, 1190–1199, 2002.

[1] S. Rhea, D. Geels, T. Roscoe, J. Kubiatowicz.Handling Churn in a DHT. In *Proc. USENIX Annual Techical Conference*, 127–140, Boston, MA, USA, June 2004.

[RD01] A. Rowstron, P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large Scale Peer-to-Peer Systems. In *Proc. IFIP/ACM Int.l Conf. on Distributed System Platforms (Middleware)*, 329– 350, 2001.

[SETI] SETI@Home Project. setiathome.berkeley.edu

[SS05] S. Sodhi, J. Subhlok. Automatic Construction and Evaluation of Performance Skeletons. In *Proc. 19th IEEE IPDPS*, 88–97, 2005.

[SGL04] M. Srivatsa, B. Gedik, L. Liu. Scaling Unstructured Peer-to-Peer Networks With Multi-Tier Capacity-Aware Overlay Topoloies. In *Proc. 10th Int.l Conference on Parallel and Distributed Systems*, 17–24, 2004.

[SM+01] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnam. Chord: a Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM*, 149–160, 2001.

[TD05] G. Tsouloupas, M D. Dikaiakos. GridBench: A Workbench for Grid Benchmarking. In *Advances in Grid Computing -EGC 2005*, 211-225, 2005