

ONSP: Parallel Overlay Network Simulation Platform¹

Yinghui Wu, Ming Li, Weimin Zheng

Department of Computer Science and Technology, Tsinghua University, Beijing, China
{wuyinghui97, lim01}@mails.tsinghua.edu.cn, zwm-dcs@tsinghua.edu.cn

Abstract. In research of overlay networks, simulator takes a very important role. However, popular simulators, such as ns and PlanetLab can't meet the scale and performance requirement of overlay research. Although it is necessary for overlay researchers to observe activities of million nodes, current simulators can not give such simulation result. To improve the overlay network research efficiency, we design and implement ONSP, a novel parallel overlay network simulation platform, which provides parallel discrete event simulation of overlay networks on high performance cluster. With this tool, we are able to build overlay network simulator in large scale easily and test it in short time. The test result proves that ONSP can well address the requirement of performance and scalability.

Keywords

overlay network, discrete event simulation, parallel

1. Introduction

Recently, the lack of suitable simulator greatly encumbers the research of overlay networks. Overlay network is the application-layer overlay on top of the existing internet routing substrate. This layer conceals the details of internet routing, and helps the application on top of it identify and communicate with each other in a large scale. Verifying the protocols prior to their deployment is a fundamental step in research of overlay network protocols. Because the overlay network tends to be large, heterogeneous system with complex interaction between a large numbers of internet nodes, testing it in a real environment is nearly impossible. Most of the time, we employ a network simulator to evaluate the overlay network protocol or application in a controlled environment. There are some simulation tools or environments, such as ns and PlanetLab [4], and they work well on a small scale. But with the advance of overlay network research, protocols today sometimes deal with nearly one million internet nodes, and it is impossible to use the old simulators on such a large scale.

¹ This research is supported by The National High Technology Research and Development Program of China (G2001AA111010) and Chinese National Basic Research Priority Program (G1999032702)

PlanetLab is itself a small scale live network environment. The simulator ns addresses more details about the network layer, so it faces scalability limitation beyond a few hundred nodes. So the research of new large scale simulator for overlay network has attracted a lot of interests recently.

Figure 1 depicts a generic overlay network simulator in our opinion. The simulate-engine is the core of simulation, which drives the execution of the simulator. Design and implementation of this layer determines the performance of whole simulation system. The upper layers are user interface, designed for the convenience of protocol implementation. It could be simply the API interface exported from simulate-engine, such as simulator included in FreePastry [8]; or it could be a complete framework, such as MACEDON [1]. How these layers are designed determines the usability of this simulator. Currently, a lot of work has been done on the upper layers, but little attention has been paid to the simulate-engine itself, so even though new simulators accelerate the implementation of special protocol, they haven't eventually eliminate the limitation of scale and performance.

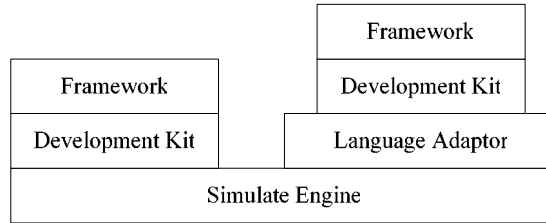


Fig. 1. Layers of a generic overlay network simulator

To address this problem, we plan to build our overlay network research environment from the bottom in two steps: (1) build a high performance simulator with coarse interface as the simulate-engine; (2) build complete development framework based on the simulator. ONSP is production of the first step. It is a parallel event driven simulator, which utilizes the parallel nature of high performance cluster to address the limitation of performance and scalability. A parallel protocol is carefully designed to improve the parallelism and make use of all the available resource in the cluster. As a simulator built for overlay network evaluation, it has some extra advantages:

- ONSP omits the low level details of internet routing substrate, because overlay networks conceal the details and pay little attention on them. This simplifies the implementation and greatly improves the performance.
- The parallel protocol and synchronization strategy is carefully designed. With the increment of resources in cluster, this platform can serve larger scale and achieve higher performance.
- ONSP can use most of the topologies, and it is specially optimized for widely used transit-stub topology [2] [3].

With the test result of ONSP, we believe we have well addressed the limitation of performance and scalability in overlay network simulation. For the convenience of protocol implementation, we also include some high level components in ONSP now,

but they are far from enough; we will build a complete overlay network research environment on ONSP at the second step.

The rest of this paper is organized as follows. We first introduce parallel discrete event simulation in section 2. Then, we detail our design in section 3. In section 4 we give the experimental results, and then we conclude in section 5.

2. Overview of and Parallel Discrete Event Simulation

Parallel discrete event simulation refers to the execution of single discrete event simulation program in parallel computers [9]. A discrete event simulation model assumes the system simulated only changes state at discrete points in simulated time. The simulation model jumps from state to state upon the occurrence of events. In a serial simulation system, there is a global clock and all the events are executed in serial order of timestamp.

Serial discrete event simulation suffers from the limitation of performance, so for a long time, parallel discrete event simulation has attracted a considerable amount of interests. To find out the dependency of events, the states of system are first divided into some distinct subsets, which we called Logic Process (LP). Any event can only affect the states of one LP, and LPs only affect each other by sending timestamped messages. So we need not process the events in whole system one by one, instead, we only process the events in one LP in serial order. This is the fundamental rule of parallel discrete event simulation, which we called Local Casuality Constraint. Richard M. Fujimoto [9] gives the definition of Local Casuality Constraint: a discrete event simulation, consisting of logical processes (LPs) that interact exclusively by exchanging timestamped messages, obeys the local casuality constraint if and only if each LP processes events in non-decreasing timestamp order.

There are a lot of parallel simulation protocols which obey the Local Casuality Constraint, and most of them fall into two categories: conservative and optimistic. In conservative protocols, any LP processes one event only when it determines that there is no other event with a smaller timestamp could be sent to itself. Though most of the events are processed ahead of global time, the Local Casuality Constraint is preserved at any point of simulated time. On the contrary, in optimistic protocols LP can execute unsafe events ahead of time and it rollbacks when error occurs. Optimistic protocol can utilize more parallelism than conservative protocol, but it also introduces more cost of rollback. Which one of these two protocols is better is mainly determined by the given application.

When considering overlay network simulation, we prefer conservative protocol for two reasons: first, it is impossible for ONSP to rollback without the help of applications running above it when error occurs; second, we believe that with the optimization based on network topology and carefully designed synchronization strategy, conservative protocol can also offer a high performance.

3. Simulation Strategy

3.1. Discrete event simulation of overlay network

Nodes in overlay network maintain their states regarding their local activities and messages from their neighbors. The timer to schedule local activities and the messages are all fired or transmitted at discrete points of time, so any nodes only change their states at discrete points. As we have described before, such a system can be succinctly described in discrete event simulation model. In this model, messages and timers are mapped to events, and states of overlay node are mapped to LPs. Events such as message reception, completion of scheduled timers, and occurrence of user commands are sent to LPs and trigger the overlay protocol to perform protocol actions on the corresponding overlay nodes. Thus, the state of whole system is changed from one state to another. By analyzing the discrete system states and their changings, it is sufficient to capture the intricacy of overlay network.

3.2. Architecture

Figure 2 shows the architecture of ONSP. All the overlay nodes in system, which are mapped to LPs as described before, are firstly partitioned into several parts. To maintain load-balancing, the numbers of nodes in all the parts are nearly equal. Then each part is simulated on one simulator. Simulator is the manager of events. It manages local events in an event-queue, and dispatches events, such as timer events or message events to their destined LP (overlay node) and triggers the execution of overlay protocol at the specified time; during the execution of overlay application, new events are generated, some are rescheduled timer and some are messages sent to other nodes. Simulator conceals the distribution of overlay nodes, and the overlay application can send messages to any other overlay nodes by invoking the interface of simulator. Simulator collects all the generated events, insert events for local nodes into local event-queue and send events for other nodes to their destined simulator. Simulator is a standard parallel application written in C++. They communicate with each other by message-passing using MPI. Simulators are deployed to workstations. Multi simulators can be deployed to one SMP workstation to utilize multi-processors. To reduce the network latency, workstations are connected with high bandwidth, low latency networks, and here we use Gigabyte Myrinet [16].

We regard simulator as the minimal unit of parallelism; it manages all the local events and processes them in serial order. The execution order between simulators is defined by the conservative parallel protocol, which strictly obeys the Local Causality Constraint. The synchronization strategy in this protocol is carefully designed in order to reduce the synchronous ratio and improve the speedup. In the simulation, there are two assumptions to simplify the design:

- Overlay node is the minimal unit of logic. Details within node are omitted. There are no other smaller simulation entities, that is to say, it is impossible to schedule events for a hard disk, or something else.

- In overlay networks, message transferring latency holds a majority of processing time, and this is also the most important factor affecting the behavior of protocols. We pay no attention on the message processing time in each node and regard this time as zero.

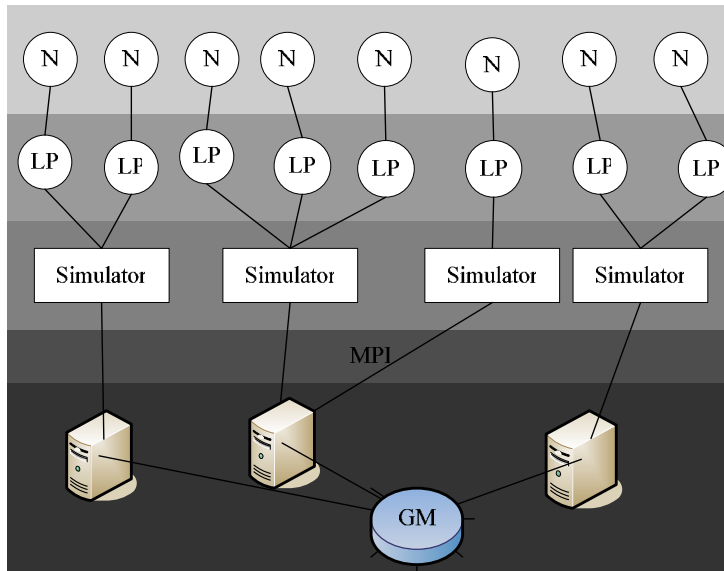


Fig. 2. Architecture of ONSP

3.3. Parallel Protocol

ONSP designs and implements a simple conservative protocol. Figure 3 depicts the working flow of simulator in this protocol. Simulators are independent when finding and processing safe events; they only communicate in the synchronization period, sending collected events and exchanging clock information.

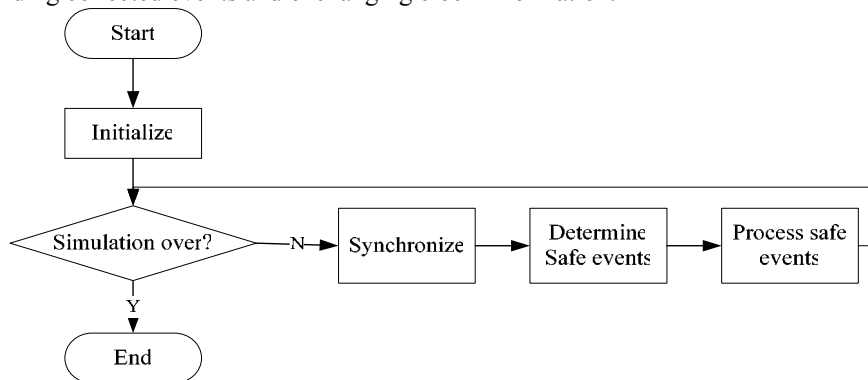


Fig. 3. Working flow of simulator in parallel protocol

The key problem in conservative protocols is to determine the safe events. Before explaining this in detail, we first present a prerequisite restrict on topology: the network topology should obey the “triangle inequation”:

$$\forall a, b, c \in V, \text{latency}(a \rightarrow b) + \text{latency}(b \rightarrow c) > \text{latency}(a \rightarrow c) \quad (1)$$

With this restrict, we can predict at which time one node could affect other nodes from its next execution point and network latency. “Triangle inequation” is tenable most of the time in internet routing, so this restrict does little harm to the evaluation of overlay networks.

Then we explain the way how simulator determines the safe events. In ONSP, each simulator maintains a moving “safe window” of time. All the events with timestamp falling into this window are believed to be safe, that is to say, the simulator will never receive other events which could fall into this window. The lower bound of the window indicates when the simulator should start event processing from. It is defined as the local simulation time of the simulator. Below is the definition of local simulation time of simulator S_i , in which $T(e)$ stands for the timestamp of event e :

$$LST(S_i) = \min\{T(e_j)\}, e_j \in S_i \quad (2)$$

The upper bound of the window indicates when the simulator could be affected by other new events and must stop to synchronize. To give the definition of upper bound, we first define the latency between two simulators as follow:

$$\text{latency}(S_i, S_j) = \min\{\text{latency}(LP_x, LP_y) \mid \forall LP_x \in S_i, LP_y \in S_j\} \quad (3)$$

As we have indicated, each LP in ONSP corresponds to one simulated network node, so the latency between LPs is the same as latency between network nodes. With equation 3, it is easy to calculate the inter-simulator latency given the network topology and all the nodes in each simulator.

With the definition of latency between simulators, we can give the definition of upper bound of safe window as follow, in which $\tau(S_i)$ stands for the upper window bound:

$$\tau(S_i) = \min\{LST(S_k) + \text{latency}(S_k, S_i) \mid \forall S_k \neq S_i\} \quad (4)$$

All the events before $\tau(S_i)$ are believed to be safe, the reason is: if any LP in any other simulator S_k sends events to S_i , the events could not affect the state of S_i before $\tau(S_i)$ because these events must occur after $LST(S_k)$ and they have at least transfer latency of $\text{latency}(S_k, S_i)$. So simulator S_i is free to process any events with timestamp lower than $\tau(S_i)$ without violating the Local Casuality Constraint.

Conservative protocols are often criticized to be over pessimistic. In equation 4, though S_k could affect S_i at $\tau(S_i)$, it seldom happens actually. So conservative protocols can only exploit a little part of all parallelism in the simulation system. But we think this is not a problem in large scale network simulation. The reason is that with the increment of network scale, more events will be generated in a small period, and the processor is kept busy even with small safe window.

To calculate the upper window bound, any simulator must know the local simulation time of all other simulators. The simulators gain this information by synchronization. The detail of synchronization is introduced in the next part.

3.4. Synchronization strategy

One key contribution in ONSP is the one-step-synchronization strategy. The ONSP system is running in synchronous manner during synchronization, so the performance of synchronization greatly affects the speedup. During synchronization, the simulator must finish two tasks:

1. Send outgoing events to their destined simulators;
2. Recalculate the LST after all new events have been received, then broadcast it to all other simulators;

Task 2 can not be started before finishing of task 1, because simulator can not calculate LST before all new events received. So in normal strategy, any simulator must communicate with each other twice during synchronization.

The key of one-step-synchronization is that instead of broadcasting the LST, we send some premature data along with the events to other simulators, and the LSTs of all simulators are calculated out by each simulator. We mark the local simulation time after last synchronization of simulator S_i as LST_{ia} , the minimal timestamp of all events in simulator S_i before this synchronization as LST_{ib} , and the local simulation time after this synchronization as LST_{ic} . Then, according to the definition of local simulation time, LST_{ic} can be calculated with equation below:

$$LST_{ic} = \min\{LST_{ib}, \min\{\min\{T(e_{ki})\} \mid \forall k \neq i\}\} \quad (5)$$

In this equation, $\min\{T(e_{ki})\}$ represents the minimal timestamp of all the events sent from simulator S_k to simulator S_i . Though LST_{ic} can not be determined before synchronization, LST_{ib} can be calculated by S_i and $\min\{T(e_{ki})\}$ can be calculated by S_k . All the simulators can send these values together with the events in one request to other simulators, and along with the completion of this data exchange, each simulator get enough information for calculating other simulators' LST. So the synchronization is finished in one request. This improvement greatly reduces the cost of synchronization, especially on a high latency network.

3.5. Partition policy

One key problem in parallel simulation is how to partition all the nodes among simulators. In ONSP, we partition the nodes according to the latency between nodes and try to aggregate near nodes to same simulator.

The first consideration about such a partition policy is to optimize ONSP for the locality-prone overlay network protocol. Though overlay networks conceal the detail of internet routing substrate, they often utilize the locality to accelerate message transferring. For a protocol with locality, nearer neighbor is more preferable as the next hop of message, so if ONSP partitions near nodes to the same simulator, more of the events should be sent within the simulator and less of them need to pass the

boundary of simulator. This will reduce the throughput on network and save the time of synchronization.

The second consideration concerns the performance of conservative parallel protocol. In PDES research, performance of conservative protocol is determined by the lookahead ability; in ONSP, it is represented by the upper window bound. Refer to equation 4, the upper window bound is mainly determined by the latency between simulators. Considering the whole simulation system, the parallelism exploited is related to the minimal of latencies between simulators. As we partition near nodes to same simulator, we also increase the latency between simulators. With a higher inter-simulator latency, we can reduce the synchronization frequency and improve the performance.

Though this partition policy is easy to express, it is some difficult to achieve. For a generic topology, there are some universe partition algorithms to gain high inter-simulator latency. For example, we can increase a threshold of latency continuously and congregate the nodes with smaller latency until we find a suitable partition. Unfortunately, the complexities of these generic algorithms are mostly too high for large scale network. For special topology, such as widely used transit-stub topology, there could be a simpler policy.

Transit-stub topology [2] is one type of hierarchy topology. Figure 4 shows the structure of a transit-stub graph. Nodes in transit-stub topology are categorized into two types: stub nodes and transit nodes. Stub nodes are the edge nodes and transit nodes are the router nodes. Stub nodes aggregate into stub domains; transit nodes aggregate into transit domains. Stub nodes in different domains can not directly communicate with each other and messages should pass through transit node as gateway, so the inter-domain latency should be larger than in-domain latency most of the time. It should be reasonable to partition all the stub nodes in a same domain into the same simulator, so partition on this topology can be simply done on the level of domains. Because each stub domain is at least connected to one transit node, so we can simply partition the transit nodes first and distribute the stub domains according to the transit nodes they are connected to. This policy is easy to implement and work with a quite low complexity, but it is too simple to achieve the highest latency.

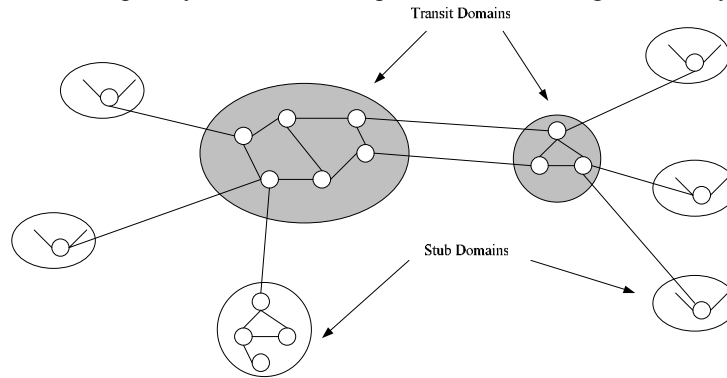


Fig. 4. Example of transit-stub topology

Increment of inter-simulator latency will accelerate the locality-prone application, but for ONSP, higher latency does not always lead to higher performance. The first reason is that in large scale network simulation, processor is not the only factor affecting the performance; memory and network bandwidth are also very important factors. With a higher inter-simulator latency, simulator can get a larger safe window and process more events before next synchronization; but at the same time, more pending events are generated and they will engage more memory and buffer on network adaptor, this could bring a heavy burden to the system and greatly increase the cost. Experiments later will prove that with higher latency, the overall performance does not improve as expected, and sometimes it drops. The second reason is that in large scale network simulation, even with a small safe window, there are still a great number of events fallen into this window, so we can still keep the processor busy and gain enough speedup upon parallel simulation. Anyway, we suggest set the latency to a proper level according to the simulation scale and available resources.

3.6. Management of Events Queue

The simulator uses a calendar queue [13] as the data structure of events queue. The calendar queue consists of a list of buckets and each bucket stands for a period of time. Events are placed into buckets according to their timestamps. There are two operations on calendar queue: insert events and pop out the earliest event. The cost of each operation is $O(1)$. Calendar queue is a high performance priority queue, and in ONSP, the period of each bucket is simply set to 1. This could be a waste of memory resource when there are no enough events in the queue, but it seldom happens in large scale network simulation.

4. Evaluation

For the limitation of time, we haven't ported any available overlay network to ONSP; instead, we use the sample application "relay", which we built along with the ONSP, to evaluate the performance. Message-passing is the fundamental operation in overlay networks, and it is simulated by this application in a simple way. The pseudo code below represents the event logic of "relay":

```

OnMessage(Message msg) {
    If (msg->hops < PROTO_HOPS) {
        neighbor = SelectNeighbor();
        msg.hops = msg.hops+1;
        Network->Transfer(msg, local, neighbor);
    }
}
OnTimer(Timer t) {
    Message msg = new Message();
    neighbor = SelectNeighbor();
    msg->hops = 0;
    Network->Transfer(msg, local, neighbor);
}

```

We also defined “ratio of locality” in this application. Locality affects the way one node selects its neighbors. The ratio of locality is defined as how much percent of their neighbors are within the same simulator.

ONSP and the sample application are implemented with C++ language. All the experiments are performed on a cluster connected by 2 Gigabytes Myrinet. Each workstation in the cluster consists of 4 700MHz Xeon CPUs and 1G memory. In this experiment, the largest topology scale we used is 100,000 nodes; this is due to the limitation of gt-itm [14], and we believe current evaluation result is sufficient to evaluate the performance of ONSP.

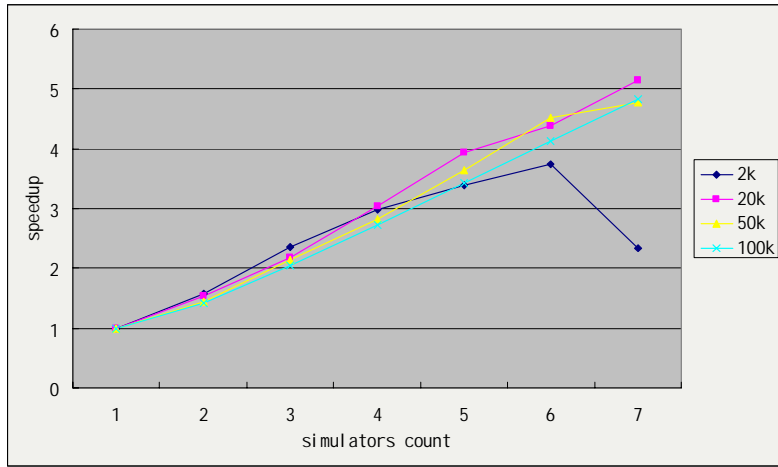


Fig. 5. Speedup of different scale overlay network simulated with different parallel simulator count.

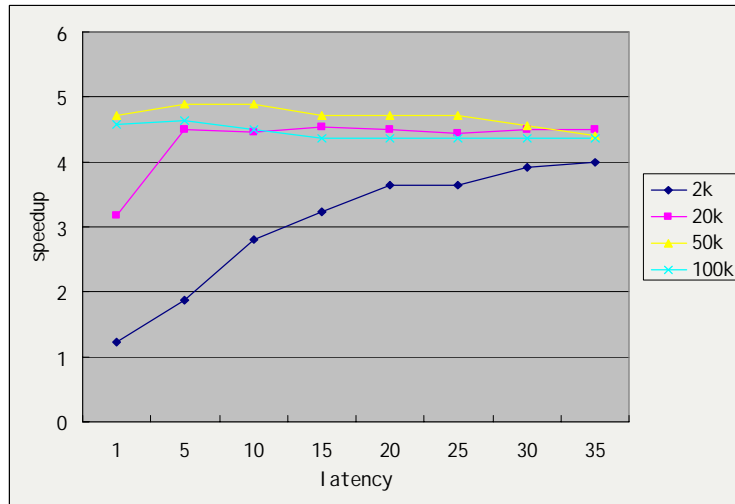


Fig. 6. Speedup of different scale overlay network simulated with different setting of latency between simulators

Figure 5 shows the speedup of different scale overlay network simulated with different simulator count. In these experiments latency between simulators are set to the max value calculated on the node partitions, and the locality ration is set to 50%. The result shows an acceptable overall performance of ONSP. The performance of 2k scale network simulated with 7 parallel simulators is one exception in Figure 5; it drops sharply. The reason of this phenomenon is the greatly drop of latency between simulators, and this is caused by the partition way of nodes. In figure 6, we plot the speedup of different scale under different setting of latency, and the curve of 2K gives the reason of sharp drop in figure 5. As we have mentioned, though increment in latency between simulators can reduce the frequency of synchronization, but it can not always improve the performance. Curves of 20K, 50K and 100K all prove this. The reason is that when one simulator continues to work for a long period, it will generates a lot of pending outgoing events, and this could lead to a great burden on memory and buffers on network adaptor. We suggest set the latency according to the scale and the available resources. In figure 7, we evaluate the performance with different locality settings. The experimental result proves that the overall performance of simulation gains obvious improvement when the application level protocols exploit more localities in topology.

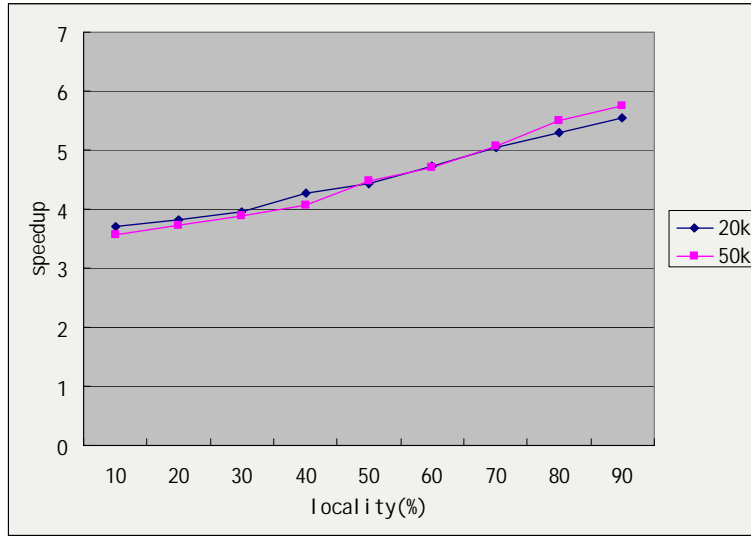


Fig. 7. Speedup of different scale network topology simulated with different locality ratio

5. Conclusion and Future work

In this paper we present ONSP as a simulation platform for large scale overlay network. As the first step in solving the problem of overlay network simulation, we pay more attention on the performance and scale ability. We make use of parallel

discrete event simulation and implement our system based on high performance cluster. The result proves that this architecture can provide an acceptable performance for large scale network simulation. Though we have provided some useful tools to accelerate the implementation of special application level protocols, it is still far from enough. In the future, we will change our focus on setting up a complete framework for overlay network study and implementation based on ONSP.

References

- [1] A.Rodriguez, C.Killian,S.Bhat, D.Kostic and A.Vahdat. *MACEDON:Methodology for Aotomatically Creating, Evaluating, and Designing Overlay Networks*. Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation, March 2004.
- [2] Ellen W. Zegura, Kenneth Calvert, and M. Jeff Donahoo. *A Quantitative Compare of Graph-Based Models for Internet Topology*. IEEE/ACM Transactions on Networking, 5(6), December 1997.
- [3] Ken Calvert, Matt Doar and Ellen W. Zegura. *Modeling Internet Topology*. IEEE Communication Magazine, June 1997.
- [4] Larry Peterson, Tom Anderson, David Cluller, and Timothy Roscoe. *A Blueprint for introducing Disruptive Technology into the Internet*. In Proceedings of ACM HotNets-I, October 2002.
- [5] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. *A Scalable Content-Addressable Network*. Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications.
- [6] Antony I. T. Rowstron and Peter Druschel. *Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems*. In Middleware, 2001.
- [7] I. Stoica, R.Morris, D. Karger, M.F.Kaashoek, and H.Balakrishnan. *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*. Technical Report TR-819, MIT, March 2001.
- [8] Rice University, *FreePastry*, 2004, <http://www.cs.rice.edu/CS/Systems/Pastry/FreePastry>
- [9] R.Fujimoto. *Parallel Discrete Event Simulation*. Communication of the ACM, 33(10): 30-53, October 1990.
- [10] D.Nicol, *The Cost of Conservative Synchronization in Parallel Discrete Event Simulation*. Journal of the ACM, vol. 40, pp. 304-333, April 1993.
- [11] L.Lamport. *Time, Clocks, and the Ordering of Events in a Distributed System*.Communication of the ACM, Vol.21, No.7, July, 1978.
- [12] Clinton Kelly, IV, Rajit Manohar. *An Event-Synchronization Protocol for Parallel Simulation of Large-Scale Wireless Networks*. Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications. October 23-25, 2003.
- [13] R.Brown. *Calendar Queues: A Fast O(1) Priority Queue Implementation for the Simulation Event Set Problem*. Communications of the ACM, 31(10), 1220-1227, 1988.
- [14] Georgia Institute of Technology. *GT-ITM*. <http://www.cc.gatech.edu/projects/gtitm/gt-itm/gt-itm.tar.gz>
- [15] *The Message Passing Interface (MPI) standard*. <http://www-unix.mcs.anl.gov/mpi/>.
- [16] Myricom, Inc. Myrinet Overview. <http://www.myri.com/myrinet/overview/index.html>