MON: On-demand Overlays for Distributed System Management *

Jin Liang, Steven Y. Ko, Indranil Gupta and Klara Nahrstedt
University of Illinois at Urbana-Champaign
{*jinliang, sko, indy, klara*}*@cs.uiuc.edu*

Abstract

This paper presents the management overlay network (MON) system that we are building and running on the PlanetLab testbed. MON is a distributed system designed to facilitate the management of large distributed applications. Toward this goal, MON builds *on-demand* overlay structures that allow users to execute *instant management commands*, such as query the current status of the application, or push software updates to all the nodes. The on-demand approach enables MON to be light-weight, requiring minimum amount of resources when no commands are executed. It also frees MON from complex failure repair mechanisms, since no overlay structure is maintained for a prolonged time. MON is currently running on more than 300 nodes on the PlanetLab. Our initial experiments on the PlanetLab show that MON has good performance, both in terms of command response time and achieved bandwidth for software push.

1 Introduction

In recent years, large distributed computing systems such as the PlanetLab [18] are increasingly being used by researchers to experiment with real world, large scale applications, including media streaming, content distribution, and DHT based applications. While a realistic environment like the PlanetLab can often provide valuable insights lacking in simulations, running an application on it has been a difficult task, due to the large scale of the system, and the various kinds of failures that can occur fairly often [8]. Thus an important tool is needed that helps application developers to *manage* their applications on such systems.

Imagine a researcher who wants to test a new application on the PlanetLab. To do this, the researcher needs to first *push* the application code to a set of selected nodes, then *start* the application on all the nodes. Once the application is running, the researcher may want to *query* the current status of the application, for example, whether the application has crashed on some nodes, and whether some error message has been printed out. Later, if a bug is identified, the researcher may want to stop the application, upload the corrected version, and start it again. To accomplish the above tasks, what the researcher needs is the ability to execute some *instant management commands* on all the selected nodes, and get the results immediately. Although many useful tools exist on the PlanetLab, such as status monitoring and query [1, 15, 4, 10, 11], resource discovery [16], and software distribution [17, 6], few of them allow users to execute instant management commands pertaining to their own applications.

PSSH [5] and vxargs [7] are two tools for executing commands on large number of machines in parallel. However, both tools use a centralized approach, where each remote machine is directly contacted by a local process. This may have scalability problems when the system becomes large, or when large amount of data needs to be transferred. The centralized approach also means there is no in-network aggregation. Thus all the execution results are returned to the local machine, even though only their aggregates are of interest.

In this paper we present the management overlay network (MON) system that we are building and running on the PlanetLab. MON facilitates the management of large distributed applications by allowing users to execute instant management commands pertaining to their applications. For scalability, MON adopts a distributed management approach. An overlay structure (e.g., a spanning tree) is used for propagating the commands to all the nodes, and for aggregating the results back.

Maintaining an overlay structure for a long time is difficult, due to the various kinds of failures that can occur in a large system. For example, if a tree overlay has

*This work was in part supported by NSF ANI grant 03-23434, NSF CAREER grant CNS-0448246 and NSF ITR grant CMS-0427089.

been created and some interior node has crashed, the tree structure must be repaired by the disconnected nodes rejoining the tree. The rejoin could become very complex, if multiple nodes fail at the same time. As a result, MON takes an *on-demand* approach. Each time a user wants to execute one or more management commands (called a *management session*), an overlay structure is dynamically created for the commands. Once the commands are finished, the overlay structure is discarded. This on-demand approach has several advantages. First, the system is simple and lightweight, since no overlay structure is maintained when no commands are executed. Second, on-demand overlays are likely to have good performance, since they are built based on the current network conditions. Long-running overlays, even if they can be correctly maintained, may have degraded performance over time. Third, since the overlays are created on-demand, different structures can be created for different tasks. For example, trees for status queries and DAGs (directed acyclic graphs) for software push.

MON is currently running on more than 300 nodes on the PlanetLab, and supports both status query (e.g., the aggregate information of different resources, the list of nodes that satisfy certain conditions, etc.) and software push commands. Our initial experiments show that MON has good performance. For a simple status query on more than 300 nodes, MON can propagate the command to all the nodes and get the results back in about 1.3 seconds on average. For a software push to 20 nodes, MON can achieve an aggregate bandwidth that is several times that an individual node can get from our local machine.

In the rest of the paper, we first present the architecture and design of MON in Section 2, then provide our evaluation results in Section 3. Section 4 provides more discussion about MON and Section 5 is the conclusion.

2 MON Architecture and Design

The MON system consists of a daemon process (called a MON server) running on each node of the distributed system. Each MON server has a three layer architecture as shown in Figure 1. The bottom layer is responsible for membership management. The middle layer is responsible for creating overlays (e.g., trees and DAGs) on-demand, using the membership information from the bottom layer. Once an overlay structure has been created, the top layer is responsible for propagating management commands down to the nodes, and aggregating the results back.

2.1 Distributed membership management

Maintaining up-to-date global membership for a large distributed system is difficult, especially when nodes

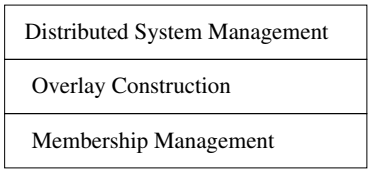


Figure 1: MON Architecture

can fail and recover fairly often. As a result, we adopt a gossip-style membership management. Specifically, each node maintains a partial list of the nodes in the system (called a *partial view*). Periodically, a node picks a random target from its partial view, and sends a `Ping` message to it, together with a small number of membership entries randomly selected from the partial view. A node receiving a `Ping` message will respond with a `Pong` message, which also includes some random membership entries. The `Ping` and `Pong` messages allow the nodes to learn about new nodes and to detect node failures. They also allow nodes to estimate the delay between each other. Such delay information can be used by the middle layer to construct locality aware overlays.

In order to maintain the freshness of membership entries, each entry is associated with an *age*, which estimates the time since a message is last received from the corresponding node. When the partial view is full and some entries need to be dropped, the oldest entries are dropped first.

2.2 On-demand overlay construction

On-demand overlay construction is a central component of our MON system. In this paper we consider the construction of two kinds of overlay structures, trees and directed acyclic graphs (DAGs). A tree structure is suited for instant status query, and a DAG is suited for software push. Since an overlay is created on-demand, we would like the construction algorithm to be quick and efficient, involving minimum startup delay and message overhead.

Ideally one may want to create an overlay that includes all the current live nodes (i.e., has full *coverage*). However, "all current live nodes" is a slippery term in a large dynamic system. In fact, merely counting the number of such nodes is a difficult task [14]. As a result, we are content with probabilistic node coverage and focus on quick and efficient overlay construction algorithms [1].

Tree Construction. The first algorithm we consider is *random tree construction*. To create an on-demand overlay tree, a client side software (called a MON client) sends a `Session` message to a nearby MON server. Each node (MON server) that receives a `Session` message for the first time will respond with a `SessionOK` message and become a child of the `Session` sender. It

also randomly picks k nodes from its partial view, and send the `Session` message to these nodes. k is called the fanout of the overlay and is specified in the `Session` message. If a node receives a `Session` message for a second time, it will respond with a `Prune` message. It has been shown that assuming the partial views represent uniform sampling of the system, such tree construction will cover all the nodes with high probability, if $k = \Omega(\log N)$, where N is the total number of nodes in the system [12].

The random tree construction algorithm is simple and has good coverage (with sufficient fanout k). However, it is not locality aware. Therefore we have designed a second algorithm called *two stage*, which attempts to improve the locality of a tree, while still achieve high coverage. To do this, the membership layer of each node is augmented with a *local list* in addition to the partial view, which consists of nodes that are close by. Each node is also assigned a random *node id*, and the local list is divided into *left* and *right* neighbors (those with smaller and larger node ids).

The tree construction is divided into two stages. During the first several hops, each node selects its children randomly from the partial view, just like the random algorithm. The goal is to quickly spread the `Session` message to different areas of the network. In the second stage, each node first selects nodes from its local list, then from the partial view if not enough local neighbors are present. To prevent nearby nodes from mutually selecting each other as children, equal number of children are selected from the left and right neighbors.

DAG construction The above tree construction algorithms can be modified to create DAGs (directed acyclic graphs). Specifically, each node is assigned a *level* l. The level of the root node is set to 1. The level of a non-root node is 1 plus the level of its first parent. Suppose a node has set its level to l and a second `Session` message is received, it can accept the sender as an additional parent, as long as its level is smaller than l. This ensures the resulting overlay contains no loop, thus a DAG.

2.3 Instant command execution

Once an overlay structure is dynamically created, one or more management commands can be executed on it. We discuss two types of management commands: status query and software push.

Status Query All the status query commands are executed in a similar fashion. First the command is propagated down the overlay tree to all the nodes. Next the command is executed locally on each node. Finally the results from the children nodes and from the local execution are aggregated and returned to the parent. Below is a (partial) list of the status query commands that we have implemented.

- `count`
- `depth`
- `topology`
- `avg <resource>`
- `top <num> <resource>`
- `histo <resource>`
- `filter <operation>`

The first three commands return information about the overlay itself, such as the number of nodes covered, the depth of the tree/DAG, and the topology of the overlay. The next three commands return the aggregate information (e.g., average, top k, and histogram) of different resource. Currently MON supports resources such as the CPU load, free memory, disk usage, number of slices, etc. Most of these resources are obtained from the CoTop [2] server on each PlanetLab node.

The last command allows (in theory) any arbitrary operation to be executed on each node, and to return some information based on the result of the operation. We have implemented the operations that compare some resource with a threshold value. For example, `filter load G 20.0` will return the list of nodes that have a CPU load greater than 20.0. To demonstrate the utility of the command, we have also implemented a more powerful *grep* operation. For example, `filter grep <keyword> <file>` will return the list of nodes on which the keyword `<keyword>` has occurred in the file `<file>`. This *grep* operation can be used for diagnosing failures of distributed applications. In fact, we have frequently used it to see if our MON server has reported some error message on some nodes.

Software Push When running an application on a large distributed system, a user may need to push the application code to a large number of nodes from time to time. Our on-demand overlays can also be used for such software push [2]. A tree structure is unsuited for software push, because the downloading rate of a node is limited by that of its parent. Therefore, we use DAG structures for software push, so that each node can download data from multiple parents at the same time. The DAG structure also improves the failure resilience of the system, because the downloading of a node is not affected by a parent failure, as long as it has other parents.

When a node can download data from multiple parents at the same time, some kind of coordination is needed between the parents. In our MON design, we adopt a *multi-parent, receiver driven* downloading approach similar to those used in recent P2P streaming systems [19]. Figure 2 illustrates how the approach works. Suppose a node p has three parents, q_1, q_2 and q_3. The file to be downloaded is divided into blocks. Whenever a node downloads a block, it will notify its children about the new

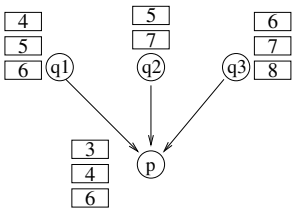


Figure 2: Multi-parent, receiver driven download

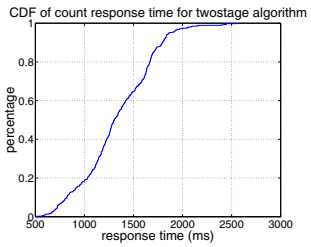


Figure 3: CDF of `count` time for twostage.

Table 1: Tree construction performance

	rand5	rand6	rand8	twostage
coverage	314.89	318.64	320.52	321.59
create time(ms)	3027.21	3035.46	2972.46	2792.03
count time(ms)	1539.19	1512.07	1369.92	1354.79

block. As a result, node p knows about the blocks that are available at each parent. Node p will then make a scheduling decision, and request different blocks to be received from different parents. For simplicity, we have used a "first fit" scheduling algorithm. For each parent, we request the first block that this parent can provide, and that is still needed by the requesting node. For example, for the scenario shown in Figure 2, node p will request block 5 from q_1, block 7 from q_2, and block 8 from q_3.

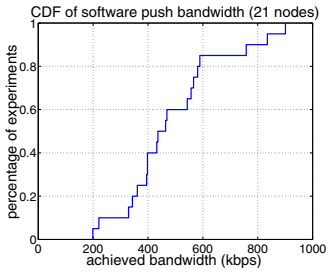
Our MON system uses UDP for most of the communication. However, for software push, we use TCP connections. Suppose a DAG has been created, a user can issue a `Push` command to push a file to all the nodes. Each MON server that receives a `Push` message from a parent will first establish a TCP connection to the parent, then create a TCP server socket to serve its children (if the server socket has not been created). Finally it will send the `Push` message to all of its children. Once all the children have established TCP connections to a node, it begins to request blocks from its parents, and advertise the downloaded blocks to its children.

3 Evaluation Results

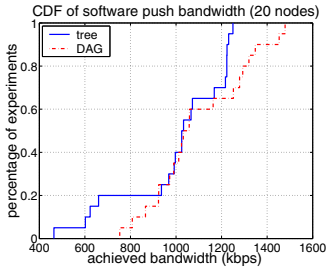
Our MON has been implemented and running on the PlanetLab for several months. The current deployment includes about 330 nodes. We have also created a web interface for people to try out MON [3]. In this section, we present some initial experiment results to evaluate our tree construction and software push algorithms.

Table 1 shows the performance of different tree construction algorithms. The experiments are conducted on our current MON deployment. For each algorithm, we create about 200 overlays and compute the average of the number of nodes covered (coverage), tree creation time, and the `count` response time. *rand5, rand6* and *rand8* are the random algorithm with $k = 5, 6$ and 8, respectively. *twostage* is the two stage algorithm with $k = 5$. The tables shows that the two stage algorithm has better performance compared with random tree construction. On average, the two stage algorithm can create a tree in less than 3 seconds and cover about 321 nodes [3]. And a simple `count` query takes about 1.3 seconds. In comparison, $rand5$ covers only about 315 nodes, and its `count` response time is more than 180ms larger. Figure 3 shows the CDF of the `count` response time for the two stage algorithm. We can see that the response time is less than 1500ms about 65% of the time, and less than 2000ms about 97% of the time.

To evaluate our software push algorithm, we pick 21 PlanetLab nodes mostly from universities in North America, and push a file of 1MB to the nodes. The file is divided into 50KB blocks. We first push the file to each node directly from our local node, and measure the bandwidth achieved. The bandwidth between our local node and most of the nodes is between 1Mbps and 3Mbps. However, for one node (`planet2.cs.rochester.edu`) the bandwidth is less than 400Kbps [4]. We then create a DAG and push the file to all the nodes. We repeat the experiment for 20 times and show the CDF of the bandwidth in Figure 4(a). The figure shows that most of the time, we can achieve a bandwidth between 400Kbps and 600Kbps, and the average is about 490Kbps. Figure 4(b) shows the result when we remove `planet2.cs.rochester.edu`. We can see most of the time the bandwidth is between 900Kbps and 1.3Mbps, and the average is about 1.1Mbps. Since all the nodes are receiving the data at the same time, this means on average we can achieve an effective aggregate bandwidth of about 22Mbps, which is about 7 times the largest bandwidth that our local node can provide to an individual node (`planetlab2.cs.uiuc.edu`). The above experiments allowed each node to have a maximum of 3 parents (the actual number of parents may be smaller). Figure 4(b) also shows the result for 20 nodes when each node has at most 1 parent (i.e. trees). We can see that about 20% of the time, the bandwidth is less than 700Kbps, and on average the bandwidth is about 10% smaller than the DAG case. This shows the advantage of DAG based multi-parent downloading schemes.



(a) 21 nodes



(b) 20 nodes

Figure 4: Software push bandwidth of MON.

4 Discussions

Many useful tools have been developed to make a distributed system such as the PlanetLab easier to use. CoMon [1], Ganglia [15] and many other tools provide resource monitoring for each PlanetLab node. The Application Manager [4] can monitor the status of individual applications. SWORD [16] provides resource discovery services. And PIER [10, 11] allows SQL like queries in large scale networks. However, these systems generally do not allow a user to execute *instant management commands* pertaining their own applications. In contrast, the `filter` command of MON can potentially be used to execute any operations (e.g., shell commands) on a node, just like PSSH and vxargs. Different from these two tools, however, MON is based on a distributed overlay structure, thus it has better scalability and allows in-network aggregations [5].

Since MON makes it easier to execute simultaneous commands on large number of nodes, it is important to have built-in security mechanisms to prevent misuse of the system. Although MON currently does not have any authentication mechanism, it is relatively easy to use public key of the user for authentication, which is already available on the PlanetLab nodes. For example, each time an overlay is created on demand, the private key of the user is used to "encrypt" some information about the user, such as the slice name and IP address. A MON server will continue with the overlay construction only if it can verify the message using the slice's public key. Timestamps can be used to prevent replay of the message, and a session key can be included for the encryption and decryption of subsequent session messages. Although encryption/decryption may increase the end to end delay, we do not expect the impact to be significant.

5 Conclusion

We have presented the design and preliminary evaluation of MON, a management overlay network designed for large distributed applications. Different from existing tools, MON focuses on the ability of a user to execute *instant management commands* such as status query and software push, and builds *on-demand* overlay structures for such commands. The on-demand approach enables MON to be lightweight, failure resilient, and yet simple. Our results further demonstrate that MON has good performance, both in command response time and aggregate bandwidth for software push.

MON is an on going project and we are continuing working on it. Specifically, our software push component is not mature yet. We will improve the system and experiment on significantly larger scales. We will also explore other on-demand overlay construction algorithms, for example, those that cover a specified subset of nodes, and those that can scale to even larger networks.

References

[1] CoMon. http://comon.cs.princeton.edu/.

[2] CoTop. http://codeen.cs.princeton.edu/cotop/.

[3] MON. http://cairo.cs.uiuc.edu/mon/.

[4] Planetlab application manager. http://appmanager.berkeley.intel-research.net/.

[5] PSSH. http://www.theether.org/pssh/.

[6] Stork. http://www.cs.arizona.edu/stork/.

[7] vxargs. http://dharma.cis.upenn.edu/planetlab/vxargs/.

[8] R. Adams. Distributed system management: Planetlab incidents and management tools. PlanetLab Design Notes PDN-03-015.

[9] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth content distribution in a cooperative environment. In *SOSP'03*, 2003.

[10] B. Chun, J. Hellerstein, R. Huebsch, P. Maniatis, and T. Roscoe. Design considerations for information planes. In *WORLDS'04*, December 2004.

[11] R. Huebsch, J. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *VLDB*, 2003.

[12] A.-M. Kermarrec, L. Massoulie, and A. J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transaction on Parallel and Distributed Systems*, 14(2), February 2003.

[13] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *SOSP'03*, October 2003.

[14] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers. Decentralized schemes for size estimation in large and dynamic groups. In *IEEE Symp. Network Computing and Applications*, 2005.

[15] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30, July 2004.

[16] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Distributed resource discovery on planetlab with sword. In *WORLDS'04*, December 2004.

[17] K. Park and V. S. Pai. Deploying large file transfer on an http content distribution network. In *WORLDS'04*, December 2004.

[18] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *HotNets-I*, 2002.

[19] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. DONet: A data-driven overlay network for efficient live media streaming. In *IEEE INFOCOM'05*, Miami, FL, 2005.

Notes

[1] In practice, due to transient and permanent node failures, a user is often prepared if not all the desired nodes can be accessed.

[2] We note that previous research work including SplitStream [9], Bullet [13] and CoBlitz [17] has addressed the problem of content distribution from one node to a large number of receivers. Our goal in developing the software push component, however, is to provide a easy-to-use system that can be integrated with our status query component, so that the user can accomplish most management tasks with the MON deployment.

[3] Note although we deploy MON on about 330 nodes, the precise number of live nodes may vary during the experiment.

[4] Note the bandwidth is the "end-to-end" bandwidth that includes the initial delay for sending the `Push` message, creating TCP server sockets, and waiting for the child connections. The effect of this initial delay will become less significant for large files. For example, the bandwidth between our local node and `planet-lab2.cs.uiuc.edu` is about 3Mbps for 1MB files, and about 13Mbps for 8MB files.

[5] Aggregating the results of arbitrary operations may need some workaround. For example, instead of executing an operation on every node and return the result, we can execute the command, return the nodes on which the operation succeeded (or failed). This way only small amount of data is returned, which may be less distracting and more interesting to the user.