PIER on PlanetLab: Initial Experience and Open Problems

Ryan Huebsch, Brent Chun, and Joseph M Hellerstein

IRB-TR-03-043

November, 2003

Intel Research Berkeley

PIER on PlanetLab: Initial Experience and Open Problems

Ryan Huebsch
Univ. of California, Berkeley
huebsch@cs.berkeley.edu

Brent N. Chun
Intel Research Berkeley
bnc@intel-research.net

Joseph M. Hellerstein
Univ. of California, Berkeley
jmh@cs.berkeley.edu

Abstract

In this paper we describe our initial experiences with deploying and running PIER, a distributed query processor, on 200 nodes of the PlanetLab network testbed. Through use cases, we show that PIER is flexible and can easily incorporate new data sources without *a priori* knowledge of their schemas. We highlight of number of important features in PIER used in our use cases, including in-network aggregation. Finally, we discuss open problems that are raised by our work to date.

1 Introduction

PIER (which stands for "Peer-to-Peer Information Exchange and Retrieval") is a distributed query processor that is designed to scale to thousands or millions of nodes. PIER uses a Distributed Hash Table (DHT) as its communication substrate to help achieve scalability and reliability without sacrificing autonomy or efficiency. Although PIER is capable of serving as a generic dataflow engine, we have outfitted PIER with a library of mostly relational operators, which are based on standard database query executors. We feel that relational query processing is sufficiently expressive for many applications, and optimization of relational expressions has been well studied. PIER is extensible, allowing users to execute their own operators if the built-in library is not sufficient for a particular application. We refer the reader to a more complete description of PIER in [2].

This paper focuses mainly on our initial experiences running PIER on the PlanetLab [4] network testbed. As expected, during deployment on PlanetLab we ran into a number of unexpected obstacles. We show data collected on PlanetLab which illustrates both that PIER is working, and that its deployment raises interesting issues. Although PlanetLab is relatively small compared to our goal of scaling to millions of nodes, it presents a useful testbed for exploring the feasibility and challenges of a system like PIER.

2 PlanetLab Examples

Large-scale federated systems such as PlanetLab present a number of system management and operational challenges, many of which are amenable to distributed query processing. Network services need the ability to discover computational and networking resources of interest in a timely and reliable manner. Global system monitoring is needed to track resource usage in the system and to identify components that have failed. Finally, distributed anomaly detection may be desirable both to detect incoming attacks against the infrastructure and to detect misuse of federated resources by users (e.g., launching a DDoS attack). While centralized solutions are effective at tackling these problems on a moderate scale, we believe that distributed query systems like PIER will prove to be a more robust and scalable solution as federated systems grow to thousands of nodes spread across the wide-area.

2.1 Resource Discovery

Resource discovery is the process of binding an abstract specification of resources to a set of physical resources matching that specification. In PlanetLab, the common case for resource discovery is identifying a set of nodes on which to deploy a network service or experiment. In cluster-based systems, resource discovery for applications typically involves finding a set of machines with some minimum resource capacity (e.g., CPU speed) or finding a set of least loaded machines. In a system like PlanetLab, which targets broad-coverage network services, the set of requests is more diverse. Other examples include finding a set of nodes at geographically distinct locations, finding pairs of nodes with multiple A.S.-disjoint network paths, and finding a set of nodes which have low failure correlation.

The diversity of resource discovery requests in PlanetLab implies that no predefined schema will be sufficient to meet all requests since all needs cannot be anticipated *a priori*. PlanetLab sensors [6] are simple daemons that export arbitrary information in a standardized way via HTTP. Given sensors that export resource discovery information, for example, PIER is then able to automatically tap in to these sensors to execute higher level queries on the underlying data. A good example of a resource discovery sensor is `ganglia-proxy`. This sensor exports static node configuration information as well as per-node resource statistics (e.g., CPU load). Should more complex needs arise, all that is required is writing and deploying a new sensor. Such sensors immediately become available for querying by PIER.

2.2 System Monitoring

A second challenge in managing a large-scale federated system is system monitoring. Here, system administrators need the ability to quickly ascertain the global state of the system and identify problematic components (e.g., nodes, network paths, etc.) that require attention. For example, a recent kernel bug caused one user to have approximately 300,000 zombie processes across all of PlanetLab. The system should allow such anomalous events to be quickly identified. Towards this end, PlanetLab currently runs a number of sensors on each node that export a rich set of per-node statistics. Currently these include the following: `slicestat` (per-slice[1] CPU, memory, network bandwidth, and thread count statistics) `netflow` (per-slice IP flow statistics, e.g., source and destination addresses/ports as well as byte and packet counts), and `scout-monitor` (per-slice byte counts). Again, note that the information used to perform effective system monitoring can evolve over time without requiring any changes to PIER whatsoever.

2.3 Distributed Anomaly Detection

Finally, a third challenge for federated systems is anomaly detection. Just as high profile Internet services are probed and attacked with alarmingly regularity, so will large-scale federated testbeds once they reach a critical mass of visibility and utility. Further, given the widespread coverage of these systems, the potential damage that can be caused by a legitimate distributed program gone awry also increases substantially. Thus, it is advantageous for systems to contain some automatic distributed anomaly detection both for outgoing and incoming traffic. Towards this end, PlanetLab currently offers several sensors which can assist in anomaly detection queries. These include `netflow` and `scout-monitor` as described previously, as well as `snortsensor`, a sensor server interface to the Snort [5] intrusion detection system. Given the above, distributed anomaly detection then might proceed by continuously running aggregate queries over, say, the number of unique destination IP addresses and drilling down with more specific queries to identify the misbehaving slice.

[1]A slice is a fundamental abstraction on PlanetLab. It comprises a network of virtual machines, each of which is bound to some set of local resources. Each network service on PlanetLab is deployed in its own slice.

3 Continuous In-Network Aggregation

Aggregation is one of the most commonly used operators in system monitoring type applications. Users are generally interested in summations, maxima, or averages when querying PlanetLab. This is not coincidental. Because of the enormous amount of raw data, aggregation is extremely important in helping users determine where interesting events are occurring and the magnitude of the event. Following that, users are more likely to issue specific 'drill down' queries to determine exact causes and view detailed information.

Aggregation is both a common operation and one that involves coordinating data from all over the network. Hence we have implemented *in-network* aggregation in PIER for enhanced efficiency of this important task. To perform in-network aggregation, we first define a root node for the aggregation by randomly selecting a key in the DHT key space. The node responsible for that key will produce the result tuples and forward them to the requester.

The query is disseminated to every node that has base data (often every node in the system). Those nodes then read the raw data from the local access method and route a message via the DHT to the root node. Since the root is defined by a key, nodes are not aware of which node is actually responsible for the root. This allows the DHT to seamlessly adjust the root based on changes in the underlying network or node membership. Because nodes route data to the root via the DHT (as opposed to direct IP), PIER is able to intercept each message at each hop along the path to the root. When multiple nodes route along the DHT topology to a single root, the result is a (usually reasonably well-balanced) tree of communication along DHT edges.

At each hop, the message is passed up from the DHT layer to the query processor, which will hold the message and wait for additional messages from other nodes. After a predetermined timeout, the node aggregates all the messages it has received and sends the new partial aggregate (e.g. a running count and sum for an average query) to the next hop. Eventually the hierarchically aggregated data will reach the root where it will be aggregated with data from the rest of the network.

Because we wish to perform a continuous aggregate, each node will periodically rescan the access method and send a new message toward the root. It now becomes important to ensure that only the latest data is used to calculate the current aggregate; old, stale data should be removed from the calculation. On the other hand, if a node is unresponsive for a short period of time, we would like to use its last known value as a substitute for the current unknown value. To do this we cache raw data and partial results at each node along the path to the root. When a new value is received, the cache will replace the old value. If a new value is not received with in a pre-determined timeout, the stale value is removed from the cache. The nodes now generate partial results using data that exists in the cache.

This method is very similar to soft-state maintenance of data. Soft-state requires the periodic renewal of data in order to compensate for failures. Our problem is slightly different in that the aggregate computations depend on the soft state being stored or used exactly once – duplicate copies will often produce an incorrect answer, whereas in many traditional uses of soft-state that is not an issue. Second we expect the data to change over time, so updates serve not only to renew data, but to update it as well. This means that the interval between updates is data-dependent, rather than simply time-dependent.

A number of inconsistencies could occur with this method. First, data used to produce a result aggregate may have been collected at different times. We believe that for most applications this will be sufficient. If stronger semantics are required, an epoch number could be tagged to the partial results. This is very similar to the way aggregation in sensor network was done in [3].

A second problem exists when a path to the root changes during execution. In this case two nodes may be counting data received from a child in the tree. One node, the previous parent, will be including cached data till it expires, and the new parent will be receiving fresh data. This creates a trade off in the accuracy of the result and the resilience to temporary communication failures.

We are currently engaged in improving our aggregation, which includes work on understanding the structure of the tree, resilience to failures, and increased efficiency.

4 Experiences and Open Problems

PIER was written completely in Java. The code base has approximately 11,000 lines of PIER code and 4500 lines of supporting code (excluding the DHT). For network communication we use an asynchronous UDP protocol with acknowledgments and retries similar to TCP however there is no connection setup overhead.

We use Bamboo for the DHT layer [1]. Bamboo is a relatively new DHT that is designed to be extremely reliable, especially in situations of high node churn. PIER's architecture is designed to be DHT-agnostic, and in the past has run over Chord and CAN as well as Bamboo. We expect that there are performance trade offs between the various DHT algorithms, a topic we hop to explore in future work.

4.1 Simple Queries

To perform our experiments, we deployed PIER on approximately 210 nodes on PlanetLab. These were all the nodes we could access at the time we ran our tests excluding the nodes only connected on Internet2. Nodes only on Internet2 can only communicate with other nodes on Internet2. However, there are substantial number of nodes on PlanetLab only on the commodity Internet.

We started by running a simple test query. On every node executing the query a tuple containing the IP address of the machine is generated and sent directly to the node issuing the query (not a PlanetLab node). It takes approximately 15 seconds for the query to disseminate to all nodes and the responses to be received.

During query dissemination we noticed an usually high amount of network traffic. This was due to a bug in our query dissemination code, however as the time we wrote this paper the bug had not been located. We believe that once this bug is located, the time to execute the query will be faster.

After running several other queries we noticed that not every node was processing the query. We went back to our simple query used above, except this time the query's physical network payload was enlarged. We noticed that this version of the simple query also suffered poor recall. After further testing it was concluded that queries larger than 1200 bytes (excluding some message overheads) were not being received by all nodes. We suspect this due to the query message being fragmented by IP and then being dropped by some nodes (or probably firewalls in front of those nodes).

Even if only a few nodes are unable to process these packets more nodes will be transitively affected by the DHT overlay, which will try to route messages through the unfriendly nodes. This would explain the drop from 210 responding nodes to 142.

4.2 In-network Aggregation Queries

Our next goal was to run a simple continuous in-network aggregation query. For this we generated the same one tuple per node as above and had PIER aggregate the count of those tuples in the network using the method described earlier. Figure 1 shows the count reported by PIER with respect to time for a 15 minute period.

It takes PIER about twenty seconds till the result is nearly correct. Other systems currently used on on PlanetLab for similar monitoring take on the order of minutes to produce the same results. The curve is not smooth due to the DHT adjusting its routing tables. Recall that the aggregation algorithm routes a message via the DHT toward a root key. As it is aggregated along that path, the value is cached for a short period of time in case the next message is delayed. However when the route changes, the value will be simultaneously counted by two nodes till the value expires from the cache. Because the DHT is sensitive to small changes in latency, this route flapping could result in small jumps in the graph.

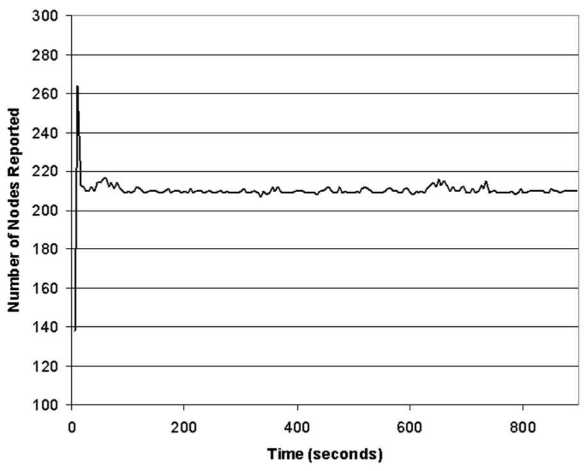

Figure 1: Continuous Count of Nodes

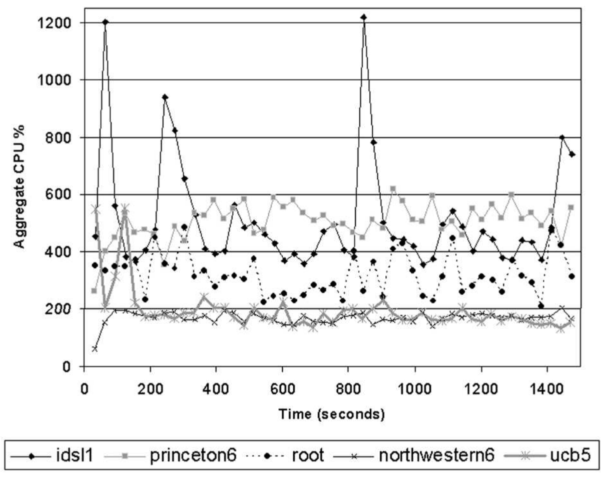

Figure 2: Aggregate CPU usage over time for the top five slices on PlanetLab.

The magnitude of the jump is dependent on where in the tree the route changes. Changes low in the tree are likely to only cause an increase of one or two nodes, while changes high in the tree can cause larger spikes.

4.3 Aggregating Slice Information

Our next query is a continuous in-network aggregation of data retrieved from the slicestat sensor. The query calculate the overall usage of resources (CPU, memory, send/receive bandwidth, and number of tasks) for a particular slice over all of PlanetLab. This query was larger than the 1200 byte limit for some nodes, so the results are calculated from about 140 nodes.

Figure 2 shows aggregate CPU usage per slice over all the nodes reporting information. Only the top five slices are shown for clarity. Unfortunately this graph shows a large amount of variance between samples. Some of this variance is real changes in system usage, while some of it is due to three factors described below.

First, with some nodes not being able to process the query due to the size of the query (for the above mentioned fragmentation problem), the network was more unstable. We confirmed this by running a larger version of the continuous node count query and noticed an increased variance in the count.

Second, the same route flapping problem impacts these numbers as well. However, a large cause for the variance and for the slow startup time is due to the slicestat sensor. On some heavily loaded nodes, the sensor took over 20 seconds to report a complete set of data. This meant we had to slow the sample rate to thirty seconds (rather than five seconds for node count queries). This, in turn, raises questions on how to set appropriate timeouts for large-scale aggregation queries given varying node CPU loads, sensor response times, and network congestion.

The large spikes shown for slice idsl1 demonstrates PIER's ability to perform continuous in-network aggregation at a sufficiently high query rate to observe this slice's bursty CPU behavior over time. Other slices may have been changing as well. However, we did not investigate them specifically. This shows that even though the numbers are not stable, the system was performing well enough to show true changes in the system.

4.4 Other sensors

Our final set of tests on PlanetLab were to see the ease and flexibility of using other data sources available to us.

Our sensor scanner is capable of processing data from all those sensors into tuples without *a priori* knowledge of the sensor or its schema. The query specifies the port and request string as well as the type of each field that is expected. Without any modifications to the scanner we were able to run a query over data from Snort. Table 1 shows the top ten Snort rules that were activated over all the nodes running the Snort sensor.

This demonstrates the ease in which new data sources can be queried by PIER without any changes to the infrastructure. This is an important property for any system deployed in the Internet where protocols, applications, and data sources are constantly changing.

Rule	Rule Description	Hits
1322	BAD-TRAFFIC bad frag bits	465,770
2189	BAD TRAFFIC IP Proto 103 (PIM)	123,558
1923	RPC portmap proxy attempt UDP	31,491
1444	TFTP Get	21,944
1917	SCAN UPnP service discover attempt	17,565
1384	MISC UPnP malformed advertisement	14,052
1321	BAD-TRAFFIC 0 ttl	10,115
1852	WEB-MISC robots.txt access	10,094
1411	SNMP public access udp	7,778
895	WEB-CGI redirect access	7,277

Table 1: The top ten rules hit by Snort over all nodes running the Snort Sensor.

5 Conclusion

The work we present in this paper opens more questions than it answers. Our relatively simple queries did not run as expected, showing that more work on the basic algorithms is still needed.

Aggregation is a prime area for future research. Achieving the correct value with real nodes that do not fail is not easy. We need to answer questions such as how to set the timeouts/refresh rates? How long should data be cached? How do we choose the root? Do we need adaptive algorithms to make these decisions? Are there more efficient ways of aggregating, i.e. do we need to send as much data each round or limit it to just data that changes? How do we handle changing routes without over/under counting values?

References

[1] Bamboo. *Submitted for publication*, 2003.

[2] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *Proc. of VLDB 2003*, Sept. 2003.

[3] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation service for ad-hoc sensor networks. In *Proc. of OSDI 2002*, Dec. 2002.

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

[5] M. Roesch. Snort – lightweight intrusion detection for networks. In *Proc. of LISA 1999*, Nov. 1999.

[6] T. Roscoe, L. Peterson, S. Karlin, and M. Wawrzoniak. A simple common sensor interface for planetlab, March 2003. PDN-03-010.