Distributed Resource Discovery on PlanetLab with SWORD

David Oppenheimer[†], Jeannie Albrecht[‡], David Patterson[†], and Amin Vahdat[‡]

[†]EECS Computer Science Division
University of California Berkeley
{davidopp,patterson}@cs.berkeley.edu

[‡]Department of Computer Science and Engineering
University of California San Diego
{jalbrecht,vahdat}@cs.ucsd.edu

1 Introduction and Architecture

Large-scale distributed services such as content distribution networks, peer-to-peer storage, distributed games, and scientific applications, have recently received substantial interest from both researchers and industry. At the same time, shared distributed platforms such as PlanetLab [2] and the Grid [6] have become popular environments for evaluating and deploying such applications. Assuming node and/or network characteristics on such platforms are heterogeneous, and that the user has a motivation (economic, social, or due to the performance properties of her application) to use a subset of the nodes, a practical difficulty in the use of such large-scale infrastructures centers around locating an appropriate subset of the system to host a service, computation, or experiment.

This choice of nodes may be dictated by a number of factors, depending on the application's characteristics. "Compute-intensive" applications might be particularly concerned about spare CPU, physical memory, and disk capacity on candidate nodes. "Network-intensive" applications, such as content distribution networks and security monitoring applications, might be particularly concerned about placing service instances at particular network locations—near potential users or at well-distributed locations in a topology—and on nodes with low-latency, high-bandwidth links among themselves. Other applications, such as distributed multiplayer games, may be concerned about both types of node attributes, e.g., low load for game logic processing and low latency to users for good interactive performance.

To automate this node selection process, we have built SWORD—a decentralized resource discovery service that is designed to satisfy queries over an extensible set of per-node and inter-node measurements that are relevant to deciding on which nodes of an infrastructure to place instances of distributed applications. This paper focuses on SWORD's PlanetLab deployment and the lessons we have learned from it. The key features of SWORD's operation on PlanetLab are its scalable, distributed query processor for satisfying the multi-attribute range queries that describe application resource requirements, and its ability to support queries over not just *per-node* characteristics such as load, but also over *inter-node*

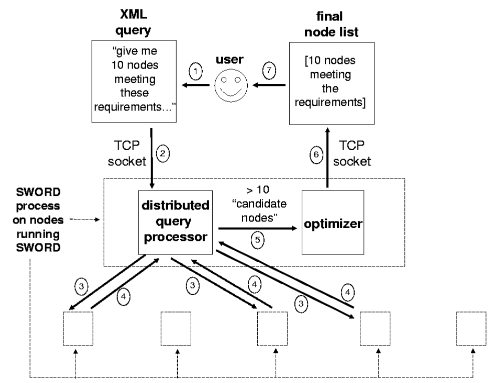

Figure 1: High-level architecture of SWORD

characteristics such as inter-node latency. Other features of SWORD are described in [11].

A SWORD user begins by specifying requirements for a set of nodes. Resources are described as a topology of interconnected groups with required intra-group, inter-group, and per-node characteristics. For example, a content distribution service for streaming media might want several "virtual clusters" of nodes, with each cluster near one portion of its geographically distributed user base. Each cluster is an equivalence class that would be composed of machines with sufficient disk space and with sufficiently low latency among nodes in each group to enable cooperative caching.

SWORD users specify a range of required and preferred values of per-node and inter-node resource measurements, with varying levels of *penalties* for selecting nodes that are within the required range but outside the preferred range. For example, the content distribution service might desire 20 ms latency or less among all nodes within each virtual cluster. However, under constraint, the service might be satisfied with latencies up to 40 ms, with correspondingly higher penalty. Latencies greater than 40 ms may be insufficient to support desired performance, corresponding to infinite penalty. SWORD endeavors to locate the lowest-penalty configuration that still meets the user's requirements.

SWORD's high-level architecture appears in Figure 1. The user writes a query expressed in XML (1) and sends

```
Group NA
  NumMachines 4
  Required Load [0, 2]
  Preferred Load [0, 1] penalty 0.01
  Required DiskFree [500, MAX]
  Preferred DiskFree [1000, MAX] penalty 5.0
  Required AllPairs Latency [0, 40]
  Preferred AllPairs Latency [0, 20] penalty 0.5
  Required OS [``Linux'']
  Required Location [``NorthAmerica'', 0, 50]
Group Europe
  NumMachines 4
  Required Load [0, 2]
  Preferred Load [0, 1] penalty 0.01
  Required DiskFree [100, MAX]
  Preferred DiskFree [1000, MAX] penalty 5.0
  Required AllPairs Latency [0, 40]
  Preferred AllPairs Latency [0, 20] penalty 0.5
  Required OS [``Linux'']
  Required Location [``Europe'', 0, 50]
InterGroup
  Required OnePair Latency NA Europe [0, 100]
  Preferred OnePair Latency NA Europe [0, 50] penalty 0.5
```

Table 1: Sample query.

it over a TCP socket to any node running SWORD (2). The query is received by the SWORD distributed query processor component on that node, which issues a distributed range query corresponding to the requirements of the requested groups (3). Once all of the results are returned from the distributed range query (4), these "candidate" nodes and their associated measurements are passed to the optimizer (5). The optimizer selects a penalty-minimizing subset of the candidate nodes and returns a list of them (along with the attribute measurements that led to their being selected) to the user (6 and 7). Note that although the optimization algorithm is not parallelized, if users employ a mechanism such as DNS round-robin to choose the SWORD entry point node for each query, then the optimization is effectively parallelized on a per-query basis.

1.1 Query Format

A sample query appears in Table 1, an extension of our earlier example. For clarity of presentation, we have converted SWORD's XML query syntax into a more human-readable format that structurally matches the actual XML syntax.

The first section of the SWORD query specifies constraints on single-node and inter-node (node-pair) attributes of desired groups. One set of single-node and inter-node constraints is associated with each group. All nodes within a node group have the same single-node and inter-node constraints, and the description of each node group also contains the number of nodes that should be in that group. In the example query, `Load`, `DiskFree`, and `Latency` are floating point attributes, `OS` is a string attribute, and `Location` is a network coordinate attribute. By placing requirements on these attributes, the user has requested two groups: a cluster of four machines in North America and a cluster of four machines in Europe. The machines in each group must have load less than 2.0 and the inter-node latency within each group must be less than 40 ms. The North American nodes must be less than 50 ms from a predefined network coordinate "center" for North America and have at least 500 MB of free disk space, and the European nodes must be less than 50 ms from a predefined "center" for Europe and have at least 100 MB of free disk space. The Preferred lines describe the shape of the penalty function for each attribute, a full discussion of which we omit due to space constraints.

The second section of the SWORD query specifies pairwise constraints between individual members of *different* groups. For example, our sample query specifies that there must exist at least one node in each group with at most 100 ms latency to a node in the other group.

1.2 Tracking and Querying Measurements

SWORD collects measurements from reporting nodes and stores them on a distributed set of server nodes. We organize these servers using the Bamboo [14] structured peer-to-peer overlay network, although essentially any structured peer-to-peer overlay network could be used. In this paper we refer to such systems as distributed hashtables (DHTs), but SWORD uses only the key-based routing functionality. On top of the key-based routing interface we build our own soft-state distributed data repository.

A node that reports n single-node attributes A_1, A_2, ..., A_n periodically sends a tuple of all of its values for those attributes to the m DHT keys k_1, k_2, ..., k_m, with $m <= n$, where each k is computed based on the corresponding value of A at the time the measurement is sent. We call each such message a *measurement report*. Associated with each attribute is a function that maps the value of an attribute to a 160-bit DHT key. SWORD provides default mapping functions for its 54 pre-configured attributes and allows the administrator to specify new ones when a new attribute is added to the system. The mapping functions convert measured values from their native datatype and range to the range of the DHT key space. The generated key is composed of high-order "attribute bits" used to partition attributes among subsets of DHT nodes so as to bound the maximum number of nodes among which values for an attribute are spread; and low-order "value bits" and "random bits" used to spread the expected range of an attribute evenly among all nodes responsible for that attribute. A second, *active* layer of load balancing can be added to these passive techniques [3], but active load balancing is not used in our PlanetLab deployment.

Upon receiving a measurement report, a server (DHT node) stores the tuple in memory. Measurement reports time out after a configurable multiple of the measurement report interval so that information about dead nodes, and nodes that have become the responsibility of another DHT server due to nodes joining or leaving the DHT, is removed. SWORD uses a multi-attribute range search mechanism similar to Mercury's [3] to find nodes meeting the single-node requirements. Once the "candidate nodes" satisfying all single-node requirements have been returned, the querying node obtains the inter-node measurements. SWORD includes an implementation of the Vivaldi network coordinates algorithm [5], and inter-node latencies are computed from the network coordinates of the returned nodes. Finally the single-node and inter-node measurements are sent to the querying node's optimizer, which attempts to find a penalty-minimizing assignment of candidate nodes to groups. The number of candidate nodes returned from the distributed range query may be larger than the number of nodes ultimately mapped to groups; conversely, an insufficient number of nodes may meet the query's requirements in which case the request cannot be satisfied.

1.3 PlanetLab-specific Details

We have run SWORD continuously on about 200 PlanetLab nodes for several months as a publicly-accessible service. To issue a SWORD query, users can download and run a simple C client from the SWORD web page, or enter the query into a text form on that web page. Any SWORD node can receive a query from the C clients and return results to the user.

To add a new metric to be reported from a node, an administrator writes a script or program that measures the new metric and writes it to a file; this script or program is then installed as a cron job that runs periodically. Finally a line is added to the SWORD configuration file on that node, specifying the attribute's name, the name of the Java class containing its value-to-DHT-key mapping function (or the name of a default datatype to use a built-in mapping function), and the name of the data file that will contain its current value. At the next update interval (two minutes in our current PlanetLab deployment), SWORD will re-read the configuration file and incorporate the new metric into the measurement report it generates, without requiring the SWORD process to restart..

SWORD currently generates reports with 54 attributes from four data sources on each node: the ganglia [15] daemon running on each node, CoTop [12] invoked periodically on each node, Trumpet [1], and the Vivaldi implementation built into SWORD.

2 Deployment experience

This section presents preliminary performance results from our deployment of SWORD on PlanetLab, and qualitative observations we have made about that deployment. The performance results are not intended as a highly accurate measure of SWORD's performance, but primarily to argue that SWORD's performance is adequate for it to serve as a useful tool.

2.1 Feasibility Study

Our first experiment aimed to determine the performance feasibility of basing SWORD on the DHT-based range search primitive described earlier. As a baseline workload we ran SWORD on approximately 200 PlanetLab nodes on August 1, 2004. Each node sent an update containing 54 metrics every 2 minutes. We measured the query latency for a query that retrieves the full range of one attribute (the node's one-minute load average) under four configurations: 1, 40, 80, and 200 queries issued per minute across the system (evenly distributed among nodes). Each attribute was mapped to approximately six nodes; therefore six nodes were visited in satisfying the range query.

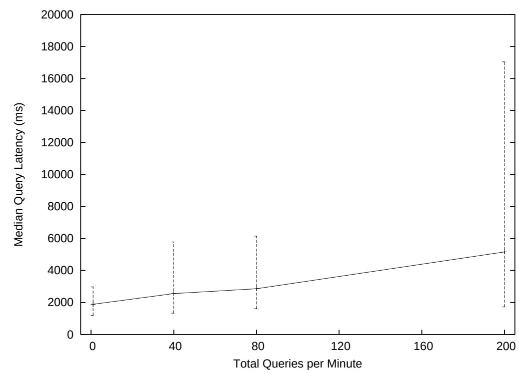

Figure 2: Performance of DHT-based range search. The bar at each x value shows median query latency and $90th$ and $10th$ percentile latency.

For comparison, we implemented a "centralized" version of SWORD in which each update is sent to one of N servers at random, and each query is sent to all N servers. We chose the servers to be machines with low load and low latency/high bandwidth network links. In contrast, the nodes used to satisfy a query in the DHT approach are selected randomly, since nodes choose their DHT IDs randomly.

Figure 2 shows the performance of the DHT range search as a function of query rate. We emphasize that these experiments were conducted over a 4-hour period during which node and network resource contention varied, and the number of nodes in the system varied slightly. These results are therefore only *approximately* repeatable. For comparison, the "centralized" version with one server offered consistently inferior median performance for query rates of 40 queries/min and above (ranging from median latency of 728 ms for 1 query/min to 12.5 seconds for 200 queries/min), while the "centralized" version with two servers offered consistently superior median performance for all query rates (with median latency ranging from 251 ms for 1 query/min to 3.3 seconds for 200 queries/min). This suggests that although DHT-based SWORD may offer acceptable performance, a non-DHT-based version with as few as two well-chosen servers may offer superior performance. The general lesson here is that users deploying services intended for the scale of

PlanetLab might be wise to consider implementing a simpler "centralized" version first, only moving to a decentralized design if they believe it will substantially improve some property of the service.

Our next experiment examined optimizer latency as a function of number of candidate nodes, for a query that requested two groups of nodes, 4 nodes in each group, with at most 150 ms inter-node latency within each group, and all nodes with a varying range of loads ranges that allowed us to control the number of candidate nodes returned. This experiment showed that even under moderately high load (the optimizer was run on a node with a load consistently in the 3-4 range, with over two dozen active slices), and with the worst case of all nodes returned by the distributed query as candidate nodes, the optimizer could satisfy the query in less than seven seconds. Still, this performance is somewhat disappointing, suggesting that we should attempt to improve the optimizer's performance and should consider moving the computation entirely to the (presumably less loaded) user's machine rather than running it on the PlanetLab node that servers as the query's entry point into SWORD.

2.2 Qualitative Observations

In this section we present several lessons learned from our deployment and operation of SWORD on PlanetLab for several months. For each observation, we have italicized the point we feel generalizes to services beyond SWORD and likely to future testbeds other than PlanetLab.

The claim has been made that DHTs are important building blocks because *a service built on top of a DHT will automatically inherit the DHT's self-configuration, self-healing, and scalability*. We found this claim to be largely true. The DHT's neighbor heartbeat mechanism and node join bootstrap protocol automatically repartition the keyspace–and hence the mapping of measurement reports to servers–when DHT nodes join or leave (voluntarily or due to failure or recovery), without the need for operator involvement or application-level heartbeats within SWORD. We benefit from the DHT's logarithmic routing scalability for sending updates, but the number of nodes touched by a range search query once it reaches the first node in the range scales linearly with the number of nodes in the DHT (assuming the queried range remains fixed). A somewhat more complex range query scheme that follows routing table pointers rather than successor set pointers provides logarithmic scaling [11] but is not currently used in our SWORD PlanetLab deployment.

Although SWORD benefits as described above from its tight integration with a DHT, this integration does cause at least one difficulty. Because PlanetLab is a shared infrastructure, the CPU load on a node can become quite high. This fact interacts badly with Bamboo's heartbeats, which declare a node unreachable if it does not respond to heartbeats within a sufficient time period. The DHT does not distinguish a heartbeat timeout caused by node or link failure from one caused by high load on the peer node. This is arguably a reasonable choice for a DHT, because nodes with extremely high loads will degrade the performance of the DHT and may therefore be best left out of the system. But it is problematic for SWORD, because we *do* want SWORD to run on very highly loaded nodes (for example, a developer might use SWORD specifically to find heavily-loaded, resource-constrained nodes). Because SWORD's measurement reporting facility is integrated with the DHT, a highly loaded node's removal from the DHT prevents it from reporting measurement updates. In our design of SWORD we aimed to treat the DHT as a "black box," not second-guessing parameters set for the DHT. A possible solution that does not require modifying or tuning the DHT, is to separate the measurement reporting functionality from the DHT and SWORD query processor. We could build a standalone reporting "stub" that runs on heavily loaded nodes that have been excluded from the DHT, and sends its measurements to a DHT node acting as a proxy (much as DHT nodes serve as a proxy for queries originating outside of SWORD). That gateway node would insert measurement reports into SWORD on behalf of non-DHT nodes, and would proxy queries and their responses on behalf of the non-DHT nodes. This would allow SWORD to accept updates and queries from all nodes without requiring the DHT and SWORD query processor to operate on heavily-loaded nodes. A generalization of this principle is that *unlinking of fate sharing between the DHT logic and logic that does not strictly require the DHT* can reduce the impact that DHT design decisions and parameters have on an application that uses the DHT.

We used the PlanetLab Application Manager [8] to automatically restart crashed SWORD instances. It was important to disable this feature during debugging, since in that setting a crashed application instance generally indicates a bug that needs to be fixed. *Automatic re-start was a mixed blessing* once we had deployed the service in "production." While it allowed SWORD to recover quickly from node reboots, and allowed us to continue to provide the service in the face of bugs, it hid transient bugs. Because periodically collecting logfiles from hundreds of machines to look for restarts is time-consuming and resource intensive, a more sensible approach is to automatically email the service operator the most recent logfile each time the application is restarted on a node. Restart allows a service to handle failure gracefully–but at times perhaps too gracefully.

PlanetLab is commonly used to test and deploy scalable wide-area services. For some services, such as monitoring, the platform is small enough that centralized solutions may offer adequate performance. It is therefore tempting to build such services with an interface to external users only at a central data aggregation point. An example of such a service is Trumpet, which collects and aggregates per-node event data. Trumpet data can be retrieved from the Trumpet server, but it is not available through a local interface on each node where data is collected. To work around this fact, each SWORD instance contacts the central server every 15 minutes to retrieve information about itself, which it then publishes along with its locally-collected ganglia, CoTop, and network coordinate measurements. The operators of the Trumpet service have indicated that they will soon be deploying a decentralized version of the service, which will make this unnecessary. But this technique generalizes to any centralized data source, at the cost of some *inefficiency in retrieving data into a decentralized system from a central server rather than from local sources*.

Finally, for the purposes of quickly turning our research prototype into a service usable by others, we found that a *simple user interface with semantics close to those used internally* by SWORD was helpful. Although our long-term vision for SWORD includes sophisticated user interfaces that allow service deployers to graphically depict desired deployment configurations and penalty functions, we first wrote a simple C client that sends an XML file from the user's disk over a network socket to SWORD. Two external users have already begun developing tools that make use of this programmatic interface. A graphical interface, a more sophisticated query language, or a SOAP interface can be layered on top of the current minimal interface.

3 Related Work and Future Work

We point the reader to closely related work in three areas: distributed range queries in DHTs [3] [13], Internet scale query processors that can be used for resource discovery [9] [17] [10], and systems for wide-area resource discovery [16] [4] [7].

We are developing a GUI to help users describe desired configurations. We are also adding a mechanism to allow users, not just administrators, to safely add new attributes to the system, so that SWORD can be used to query user application statistics and not just OS-level statistics. Finally, we plan to integrate SWORD with application deployment tools so that users can quickly deploy their application on, and periodically migrate their application to, the set of nodes that SWORD selects.

Acknowledgements

We would like to thank Sean Rhea for answering our questions about the Bamboo DHT, and Steve Czerwinski for implementing the Vivaldi network coordinates algorithm.

References

[1] Planetlab trumpet server. http://jabber.services.planet-lab.org/.

[2] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating Systems Support for Planetary-Scale Network Services. In *Proceedings of 1st Symposium on Networked Systems Design and Implementation*, 2004.

[3] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. In *Proceedings of SIGCOMM 2004*, 2004.

[4] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *HPDC '01*, 2001.

[5] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *SIGCOMM '04*, 2004.

[6] I. Foster and C. Kesselman. *The Grid 2*. Morgan Kaufmann, 2003.

[7] A.-C. Huang and P. Steenkiste. Network-sensitive service discovery. In *The Fourth USENIX Symposium on Internet Technologies and Systems*, 2003.

[8] R. Huebsch. Planetlab application manager. http://appmanager.berkeley.intel-research.net/, 2004.

[9] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proceedings of VLDB*, September 2003.

[10] S. Nath, Y. Ke, P. B. Gibbons, B. Karp, and S. Seshan. IrisNet: An Architecture for Enabling Sensor-Enriched Internet Services. Technical Report IRP-TR-03-04, Intel Research Pittsburgh, June 2003.

[11] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Scalable Wide-Area Resource Discovery. Technical Report UCB//CSD-04-1334, UC Berkeley, 2004.

[12] V. Pai. CoTop: A Slice-Based Top for PlanetLab. http://codeen.cs.princeton.edu/cotop/.

[13] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Prefix hash tree. In *PODC*, 2004.

[14] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. In *Proceedings of the USENIX Annual Technical Conference*, 2004.

[15] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler. Wide Area Cluster Monitoring with Ganglia. In *Proceedings of the IEEE Cluster 2003 Conference*, 2003.

[16] D. Spence and T. Harris. Xenosearch: Distributed resource discovery in the xenoserver open platform. In *Proceedings of HPDC*, 2003.

[17] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An Information Plane for Networked Systems. In *Proceedings of ACM HotNets-II*, November 2003.