Planeten**W**ach**H**und**N**etz: Instrumenting Infrastructure for PlanetLab *Master's Thesis Proposal*

Vitus Lorenz-Meyer
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
email vdlorenz@utep.edu

1 Introduction

We are investigating distributed parallel prefix operations. Efficient implementations, like Google's MapReduce [3] for example, utilize a reduction tree as well as involve locality-aware algorithms. In a static network reduction trees can be built once using a-priori known perfect proximity information and need never be touched again. In highly dynamic environments, such as peer-to-peer systems, this is substantially harder to achieve and maintain.
In related work two types of reduction trees for dynamic systems that both build on structured overlays, also known as Distributed Hashtables (DHTs), seem to be most prominent. Both of these structures are not concerned with exploiting existing locality. We show how these data-structures can be augmented to be more efficient and take advantage of locality information that might exist in the underlying overlay.
Distributed parallel prefix can be used to aggregate and thus reduce data from many sources. This is for example useful for statistics collection and application-level monitoring. To validate our hypothesis we are building an application-level data collection system, called ***P**laneten**W**ach**H**und**N**etz* (PlanetenWachHundNetz (PWHN) — pronounced 'pawn'), which is German that

loosely translates to 'PlanetaryWatchDogNet'. Our initial evaluation will be performed using FreePastry and Coral which is currently deployd on the PlanetLab testbed [10, 12].

2 Related Work

Work related to distributed parallel prefix can be broadly classified into the following three categories:

1. Distributed Databases,
2. Aggregation Overlays, and
3. Sensor Networks.

Distributed Databases are systems that provide semantics approximating traditional Data-Base Management Systems (DBMS), in that they are designed to allow the execution of arbitrary queries over distributed data.
Aggregation Overlays make data that is available on individual nodes available to other nodes while reducing detail (and thus size) with growing distance. This is achieved by a configurable aggregation operator. They are similar to distributed DBMS in that most allow to execute operators that do not aggregate but rather return an exact result.
Sensor Networks accomplish a slightly different goal. Sensor Networks try to present a Database-like interface for a possibly large number of distributed sensors. Sensors generate data that is kept moslty with them. A sensor network is an overlay that organizes the sensors so that they can route among themselves and providess the abilty to execute queries over the data. Consequently, Sensor Networks exhibit a large overlap with both Distributed DBMS and Aggregation Overlays. In contrast to the former two categories, however, the focus of most Sensor Networks is shifted towards resource constrained sensors.
For a more detailed overview and some examples of these categories, see [9].

3 Hypothesis and Motivation

Parallel prefix is a class of operations that are amenable to parallelization and thus distribution because of their associativity and communitivity. One application of them is log or statistics collection of distributed applications, because both tend be very homogenous and are therefore highly suitable for parallel prefix operations that extract aggregate measures. Asking every participant in a popular file-sharing network how many files he shares to extract the total number of available files is an example.
Recent systems that can be used for this purpose mostly utilize a particular implementation of an aggregation tree on top of a structured overlay. We explain how to enhance this data-structure to be efficient and locality-aware and demonstrate this in our own implementation of an application-level aggregation infrastructure for PlanetLab.

PlanetLab is a testbed for widely distributed networked applications that researchers use to test these *in vivo* on the internet. It provides central services for account creation and associated management, resource discovery and reservation.
Additional services, like more complicated resource discovery or distributed software deployment, are provided by applications that run in non-privileged Virtual Machines. The fundamental abstraction that PlanetLab builds on is the abstraction of a Virtual Machine. A researcher's account on PlanetLab is materialized as a set of virtual machines running on all individual nodes the researcher asked for. The union of all these VMs is called a *slice*, because it represents a 'slice' of the resources available to PlanetLab on all its nodes at any given time. Infrastructure services, i.e. those that provide resource discovery, reservation and slice management services, run in an administrative slice. Everything else runs in non-privileged slices.
Running experimental applications on a widely distributed set of machines naturally comprises gathering log files and statistics for debugging and testing purposes. Neither PlanetLab itself nor user contributed services provide this at the moment.

Function	Description
bool=PING()	pings the called node
STORE(key,val)	stores *val* under *key* at called node
nodes[]=FIND_NODE(ID)	returns up to k known closest nodes for *ID*
nodes[]=FIND_VAL(key)	like FIND_NODE, except returns *val* when stored at called node

Table 1: RPC Interface of Kademlia

4 Goal: Method and design

4.1 Methodology

We are building an implemention of our novel data-structure that is going to be used by an aplication-level monitoring facility for a self-organizing P2P system. This facility called PWHN runs a service on every PlanetLab node that programs running on PlanetLab can contact and request to have an aggregation tree built. This tree can then be used to route and reduce *en-route* an applications log files. By implementing a software on PlanetLab that provides this service we hope to contribute to the PlanetLab community and to researchers using PlanetLab to explore p2p.
We will implement PWHN using two different tree-like data-structures:

Firstly, a **F**inger**T**able-based **T**ree (FTT), which has been used by many related systems (for example PIER [6, 7, 11] and SDIMS [13]), exploits the tree that a DHT automatically builds by routing. The name comes from the fact that the structure of this tree depends on the live nodes' fingertables. It is the closest ancestor to our novel data-structure.
As with a FTT, our proposed data-structure utilizes a tree whose structure is directly derived from the key's relations. Unlike the FTT approach, which constructs only one global tree, our approach has the property to be able to accomondate as many trees as needed. We call our implementation **K**ey-based **M**ap**R**educe (KMR), because of its closeness to MapReduce. For a short description see subsubsection 4.2.2 and for a more elaborate explanation of it please refer to the technical report ([8]).

Function	Description
$route(key, msg, hint)$	Routes *msg* to the node whose closest to the given *key* using *hint* for the first hop
$forward(\&key, \&msg, \&nextHop)$	Function in the Application that is called on each intermediate hop while routing towards *key*
$deliver(key, msg$	Delivers a *msg* to the application on the node that is closest to the *key*
$node[]\ localLookup(key, num, safe)$	Produces a list of up to *num* nodes that can be used as a next hop for routing towards the *key*
$node[]\ neighboorSet(num)$	Returns up to *num* nodes from the local neighbor set
$node[]\ replicaSet(key, maxRank)$	Returns nodes from the replica set for *key* of nodes up to *maxRank*
$update(node, joined)$	Function in the application that is called whenever the local neighborhood changed

Table 2: KBR interface according to the Common API [2]

Both types of tree will be built using FreePastry [5], whereas only the KMR can be built on Coral [1, 4]. A FTT aggregates the data in messages while they are being routed by the DHT, and therefore needs to be notified when a messsage is forwarded by the local node. In [2] the authors argue towards a *common API* for DHTs, which includes that notification. FreePastry exports the common API and is suitable to host a FTT as well a a KMR, whereas Coral only exports the Kademlia API and can thus only accomodate a KMR. A KMR just needs to be able to find the node that owns a particular key and Kademlia exports a Find_Node RPC call.

By implementing both types of trees on FreePastry we hope to be able to quantify the advantages of our novel data-struture KMR. Building the KMR both using Coral, which makes aggresive use of locality, as well as on FreePastry should permit us to compare an intentionally locality choosing system with a mere locality-aware one.

4.2 Design

4.2.1 PWHN

PWHN is an infrstructure for a p2p system that can be used to gather, aggregate and route data from applications. Programs contact the local instance of this service to set up an aggregation tree rooted at that node. They supply executables for procurring, aggregating and evaluating data. This has the advantage that PWHN is able to collect and aggregate arbitrary data. The novel data-structure that we plan to build on is briefly described in the next section.

4.2.2 KMR

A KMR is a tree that builds on the key-based routing Layer (KBR) of structured routing overlays.
The KBR assigns unique, long (mostly 160) bit-strings (keys) to each node and provides the ability to route messages towards a certain key. All that each node on the forwarding-path of a message has to do is to guarantee to send it to another node that is closer to the requested key. Once the closest node is found, the routing algorithm terminates and the message is delivered.
The union of all possible paths (that is from all live nodes) towards a single key forms a tree - a different one for each key - this is the Finger Table-based Tree (FTT). Unfortunately, this tree is not guaranteed to be binary or balanced, and moreover does not know anything about proximity.
By a small change in the routing algorithm it can be made balanced and binary to a certain degree. Key-based routing works because of its flexibility to chose the next hop from all candidates that are closer to the key. It is this flexibility that allows the tree to be irregular. By restricting a nodes' freedom of chosing the next hop while routing to only one possible candidate, the tree of all routes can be made binary and at the same time stochastically balanced.
Furthermore, we explain how this is complemented by giving the tree awareness of its proximity space. For a more detailed description see [8].

5 Expected Results

5.1 Algorithmic

We are planning to build PWHN on top of an abstract tree interface that we will implement using a FTT as well as a KMR. Additionally, we plan to implement

this abstract tree interface on top of two different Key-based routing layers.
The first one will be the Common API exported by FreePastry [5]. FreePastry's KBR supports locality only in chosing the lowest-latency links for its routing table.
Secondly, we will implement the tree interface on top of Coral's KBR. Coral takes aggresive advantage of locality by explicitly partitioning its members into clusters that contain nodes which fall within a certain range of connection latency amongst them.
Thus, building on FreePastry as well as Coral provides us with the ability to compare an aggresively locality-aware approach with a merely locality choosing KBR. Furthermore, building a FTT as well as a KMR allows us to quantify the advantages of building a binary, stochastically balanced aggregation tree.

5.2 Broader Impact

We identified two main points:

Firstly, by implementing and testing an infrastructure for application-level monitoring we hope to be able to demonstrate the use of our novel data-struture. This infrastructure provides a blueprint for efficient monitoring, that can be used by developers to implement their own.
Secondly, by building PWHN we hope to help a researcher to set up experiments quicker when using PlanetLab. Instead of having to take care of setting up her own possibly inefficient application-level monitoring or aggregation facility, a researcher can use PWHN. It provides a guaranteedly efficient, locality-aware aggregation tree. By providing a central service for PlanetLab we will be able to contribute something substantial to the PlanetLab community.

6 Future Work

Stronger security is a major point that future versions should include. Robustness to byzantine failures, which is closely related to security, is an area that currently draws much interest in the p2p community and needs to be addressed. Another point is supplying more operators that work on XML Data and handle common operations with tuples.

7 Evaluation

To inspect the system itself, we are going to use a tree build by PWHN itself to collect and aggregate its logs. We will be using PWHN to instrument and inspect Coral. This will allow us to suggest useful metrics, collect and publish them, which will hopefully encourage others to use PWHN as well.

7.1 Locality

We are going to compare PWHN on Coral to PWHN on FreePastry in terms of speed and generated traffic, thereby quantifying the usefulness of aggresive locality-awareness. Specifically, we are going to look at

- The structure of the tree,
- latency of links used versus their number, and
- the total time for one iteration.

7.2 KMR

We will compare the effectiveness of using a KMR to using a mere FTT. This will allow us to quantify the advantages of binarity and balancedness in an aggregation tree. Specifically, we are going to look at

- The structure (fan-in) of the tree that is build, and
- the total time for one iteration.

8 Status and Timeplan

We are currently implementing the generic tree interface and the FTT on top of FreePastry. The time over the summer will be used to get this initial version running as an aggregation aid and test it using Coral's logs. Additionally, over the summer we plan to implement the KMR on the tree interface. We will also implement the KMR on top of Coral. The beginning of the fall semester will see us testing the three different results and finishing the thesis. I plan to defend at the end of the month September.

9 Deliverables

1. The source code of PWHN that is written mostly in Java, and
2. the finished written Thesis.

References

[1] CoralCDN. http://www.coralcdn.org/. Valid on 13.9.2006. 5

[2] F. Dabek, B. Zhao, P. Druschel, and I. Stoica. Towards a common API for structured peer-to-peer overlays, Feb. 2003. URL citeseer.ist.psu.edu/dabek03towards.html. 5

[3] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI)*, San Francisco, Calif., Dec. 2004. 1

[4] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with coral. In *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, Mar. 2004. 5

[5] FreePastry. http://www.freepastry.org/. Valid on 13.9.2006. 5, 7

[6] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. Loo, S. Shenker, and I. Stoica. Querying the internet with PIER. In *Proceedings of 19th International Conference on Very Large Databases (VLDB)*, Sept. 2003. URL citeseer.ist.psu.edu/huebsch03querying.html. 4

[7] R. Huebsch, B. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi. The architecture of PIER: an internet-scale query processor. In *The Second Biennial Conference on Innovative Data Systems Research (CIDR)*, Jan. 2005. URL citeseer.ist.psu.edu/huebsch05architecture.html. 4

[8] V. Lorenz-Meyer and E. Freudenthal. The quest for the holy grail of aggregation trees: Exploring prefix overlay(d) treasure-maps. Technical report, UTEP, Texas, Mar. 2006. URL http://rlab.cs.utep.edu/~vitus. Unpublished paper. 4, 6

[9] V. Lorenz-Meyer and E. Freudenthal. The field of distributed query executing, data aggregating and system monitoring. Unpublished, Mar. 2006. URL http://rlab.cs.utep.edu/~vitus. Unpublished paper. 2

[10] L. Peterson, D. Culler, T. Anderson, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, New Jersey, Oct. 2002. URL citeseer.ist.psu.edu/peterson02blueprint.html. 2

[11] PIER. http://pier.cs.berkeley.edu/. Valid on 13.9.2006. 4

[12] PlanetLab. http://www.planet-lab.org. Valid on 13.9.2006. 2

[13] P. Yalagandula and M. Dahlin. A scalable distributed information management system, 2003. URL citeseer.ist.psu.edu/yalagandula03scalable.html. 4

A

ACL	Access Control List
ADHT	Autonomous DHT
ADT	Abstract Data Type
AO	Aggregation Overlay
API	Application Programming Interface

C

CDN	Content Distribution Network
C.S.	Computer Science

D

DBMS	Database Management System
DHT	Distributed Hash Table (see **??**)
DNS	Domain Name System
DSHT	Distributed Sloppy Hashtable

F

FC	Fedora Core
FTP	File Transfer Protocol
FTT	FingerTable-based Tree (see **??**)

G

GUI	Graphical User Interface
I	
ICMP	Internet Control and Message Protocol
IDL	Interface Definition Language
IP	Internet Protocol
ISEP	International Student Exchange Program
K	
KBR	Key-Based Routing layer (see **??**)
KBT	Key-Based Tree (see **??**)
KMR	Key-based MapReduce (see **??**)
L	
LAN	Local Area Network
LGPL	Lesser General Public License
M	
MAC	Media Access Control
MB	MegaByte
MIB	Management Information Base
N	
NAT	Network Address Translation
NMSU	New Mexico State University
O	
OA	Organizing Agent
ONC	Open Network Computing (see **??**)
OS	Operating System
P	
P2P	Peer-to-Peer
PIER	P2P Information Exchange & Retrieval
PLC	PlanetLab Central (see **??**)

PLMR	PlanetLab-MapReduce
PWHN	PlanetenWachHundNetz
R	
RADC	Research and Academic Data Center
RFC	Request for Comments
RO	Routing Overlay
RPC	Remote Procedure Call
RPM	Redhat Package Manager
RTT	Round Trip Time
S	
SA	Sensing Agent
SDIMS	Scalable Distributed Information Management System
SNMP	Simple Network Management Protocol
SOMO	Self-Organized Metadata Overlay
SQL	Structured Query Language
SSH	Secure SHell
T	
TAG	Tiny AGgregation service
TCP	Transmission Control Protocol
TLD	Top Level Domain
TTL	Time-To-Live
U	
UDP	Universal Datagram Protocol
UML	Unified Modeling Language
URL	Uniform Resource Locator
UTEP	University of Texas at El Paso
V	
VM	Virtual Machine

X

XDR	eXternal Data Representation
XML	eXtensible Markup Language