The Field of distributed Query Executing, Data Aggregating and System Monitoring

Vitus Lorenz-Meyer and Eric Freudenthal

Department of Computer Science

University of Texas at El Paso

El Paso, TX 79968, USA

emails {vdlorenz, efreudenthal}@utep.edu

Abstract

Peer-to-Peer (P2P) systems have been used for a variety of purposes including monitoring of large, distributed systems. These monitoring systems have a range of features. Some aggregate data to be scalable, others strive to be as versatile as possible by providing a complete SQL-like execution engine, and still others just pass data around. Some use Distributed Hash Table (DHT) as a routing abstraction, some build trees (sometimes on-top of a DHT), and others just implement a simple gossip style protocol to propagate data.

This paper motivates a research proposal for a distributed application-level monitoring infrastructure. First it gives an overview of concepts used in related work and describes the different systems themselves.

1 Introduction

P2P systems have had a steady increase in popularity over the last few years amongst both researchers and internet users alike. While the latter group is only interested in obtaining a particular file, the former is more interested in P2Ps scientific beauty and its ability to solve old problems more efficiently.

These problems include Service Discovery, scalable Multi- and Any-Cast (CAST), group management

(GROUP), routing, Distributed Object Locating and Routing (DOLR), distributing data (in acDBMS for example), distributing load (search/query execution in these Database Management System (DBMS)), managing a large collection of distributed sensors (data sources) known as SensorNets, and monitoring of any of these systems. While a lot of effort is being put into satisfying the end-user clientelè, by programming good file-sharing tools, a large community of researchers has lately been forming around solving systems problems with the help of P2P. The Java based JXTA community [20] is one such example, the IRIS Group [40] is another.

We think that on the one hand there are a handful of P2P tools that might be useful for application-level monitoring, while on the other hand we are not aware of any good surveys about them. With this paper we strive to fill that gap and give a short but concise overview of P2P query executing, aggregating and monitoring systems before motivating our own system. We selected systems that fulfill three basic properties. They need to be designed to be

- *scalable*,
- *flexible*,
- and *robust*.

Moreover, we mention systems that fulfill these basic properties but seem to be in an early planning stage in the *related Work* Section. These properties ensure applicability to a broad range of management problems and tasks.

These systems can roughly be classified into three categories

1. Distributed Databases (subsection 3.1),
2. Aggregation Overlays (subsection 3.2),
3. and Sensor Networks (subsection 3.3).

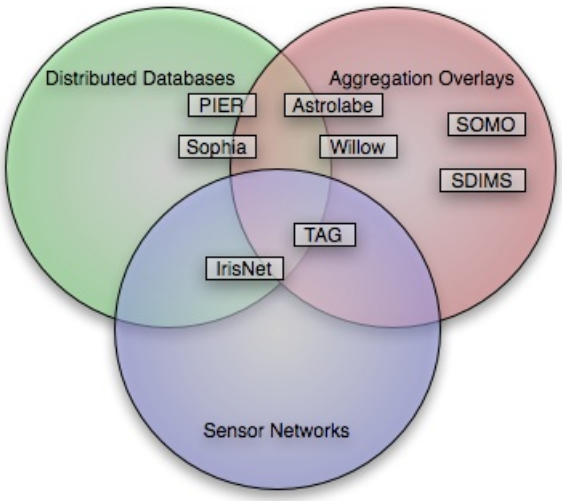

Figure 1: A rough classification of the systems in this survey into the three categories

Distributed Databases are those systems that are closest to traditional DBMS in that they are designed to allow you to execute arbitrary queries over distributed data.

Aggregation Overlays make some data that is available on individual nodes available to other nodes while reducing detail (and thus size) with growing distance. This is achieved by some configurable aggregation operator. They are close to distributed DBMS in that most of them also allow to execute operators that do not aggregate but rather return an exact result.

Sensor Networks try to accomplish a slightly different goal. Sensor Networks try to simulate a Database-like interface for a possibly large number of distributed sensors. Consequently, SensorNets exhibit a large overlap with both Distributed DBMS and Aggregation Overlays. In contrast to the former two categories, however, the focus of most SensorNets is shifted towards resource constrained sensors.

Due to this big area of intersection between these 3 classes, it is sometimes very hard to decide where to place a particular system.

After we finished this report we found ourselves with a feeling that there seems to be something missing, especially in the light of using PlanetLab [33, 35]. Thus the last section will illuminate motivations for our own solution. The reader is referred to [26] for the complete proposal.

The rest of this paper is organized as follows: Section 2 will introduce the reader to some key concepts, Section 3 explains each system in turn, Section 4 summarizes some related work, Section 5 concludes, and Section 6 illuminates our project.

2 Concepts

All these Systems have to set up some kind of overlay network above the bare P2P transport network to provide routing.

Consequently, we will call this abstraction the Routing Overlay (RO). In most cases this service is provided by a DHT [21, 28, 42, 44, 45, 48, 58]. Almost all of todays DHTs, sometimes referred to as *Prefix-based Overlays* or *Structured Overlays*, build on Plaxton et al's (commonly called PRR by people in the field) [37] groundbreaking paper about routing in unstructured networks. Their work left room for aggregation in its trees, whereas most modern implementations disregard this feature.

However some systems function perfectly well without a RO overlay. Astrolabe [51] for example does not use a DHT at all, and in fact, has no need for routing. It's epidemic gossiping protocol achieves eventual consistency by *gossiping* updates around, thus it has no need for routing, since every node is assumed to know everything about the state of the system.

Query dissemination 'down' and data aggregation 'back up' is done through the Aggregation Overlay (AO). This overlay tends to resemble its most closely related natural object, in the form of a tree. Though every system uses some kind of tree, the building algorithms as well as the actual form of these are vastly different. Astrolabe, again, does not need to explicitly build a tree on run-time, because it relies on the user specifying a name hirarchy on setup-time. Each name prefix is called a *zone* and all those nodes whose DNS name start with the same string are members of that specific zone. It builds a tree out of the hierarchy, albeit one that might have a very high branching factor (number of sub-zones to each zone and number of nodes in each leaf-zone).

A number of systems make use of the inherent structure of Prefix-based Overlays (DHTs) for building aggregation trees. The idea is as described below. Any DHT routing protocol populates a flat identifier space by assigning long bit-strings to each host. Incomplete bit-strings, that is strings with some missing or ignored bits at the end, are called *prefix* bits. Now imagine a tree in which each child's prefix is one bit longer than its parent's: A global binary tree. This assumes that the DHT fixes exactly one bit per hop. DHTs that fix more than one bit per hop will simply result in a higher branching factor. Since actual nodes will always have complete bit-strings, all internal nodes that are addressable by a prefix are 'virtual' in the sense of the tree. The physical nodes are the leafs of the tree and can be reached from the root ('empty prefix') by a unique path. We will call this structure Key-Based Tree (KBT), henceforth, because it maps the tree onto the key-space.

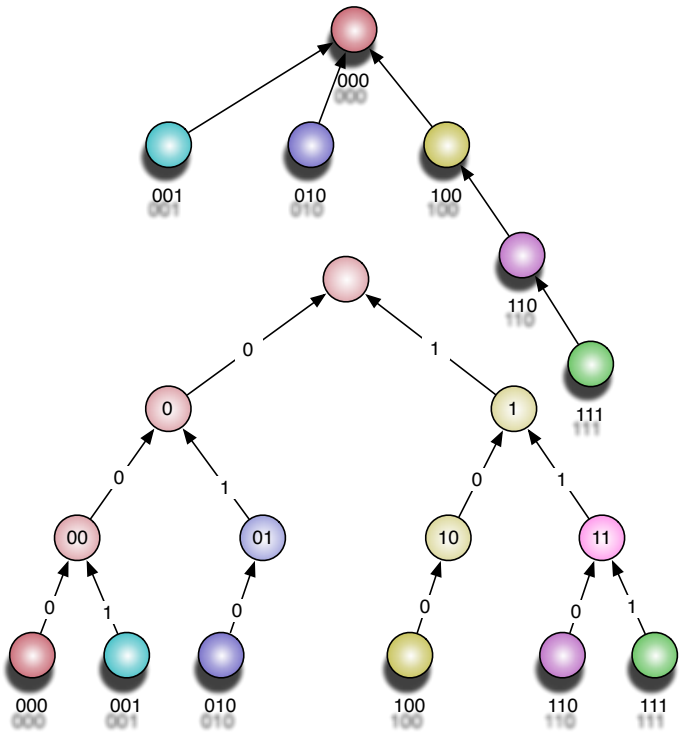

Figure 2: Sample KBT with five nodes using a 3-bit wide ID-space and a static approach in which the left child assumes the parent role ('Left-Tree'

Routing towards a specific ID in Structured Overlays works by adding at least one matching prefix bit

each hop until the node with the longest matching prefix is reached. The characteristic of DHTs that a message from every node towards a specific ID takes a slightly different path leads to the realization that the union of all these paths represents another tree - a different one for each ID. These trees are like inverted KBTs: The exact node with the longest matching prefix is the root, whereas in a KBT every node could be the root depending on its tie-breaking (see below) algorithm, since every ID matches the empty prefix. Since this data-structure depends on nodes fingertables we refer to this as a FingerTable-based Tree (FTT).

KBTs are binary if the underlying DHT fixes one bit per hop and every node will be at the same depth i, with $i = number\ of\ bits\ in\ ID$. But generally they are not balanced.

FFT!s (**FFT!**s) on the other hand will never be balanced, and moreover not all nodes will be at the same level. Moreover, they do not have to be binary. These properties make their usefulness as aggregation trees questionable. One way to fix this is described in the Proposal Section (section 6) as a possible data-structure for our project and in more detail in a technical report [25].

Some advantages of **FFT!**s over a KBT are:

- There may be as many **FFT!**s as needed, by having the root depend on the hash of some characteristic of the query, resulting in better load-balancing.
- Trees in **FFT!**s can be rooted at any node.
- Trees in **FFT!**s are kept in soft-state, thus there is no need to worry about repairing them.

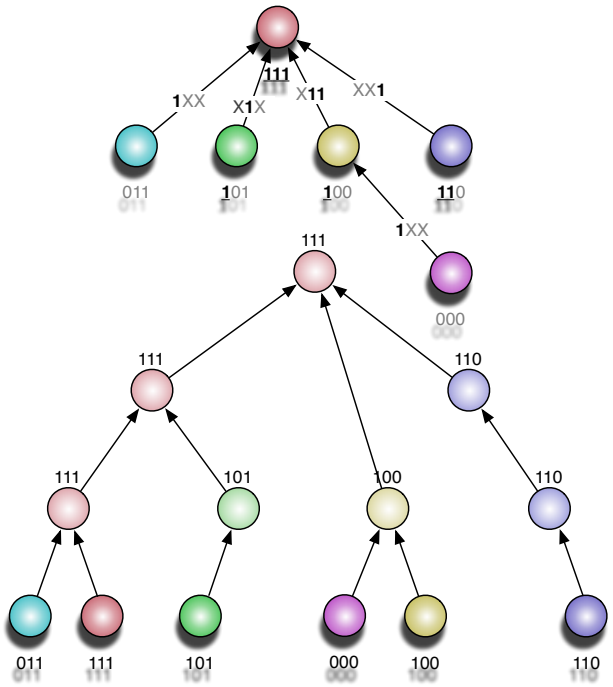

Figure 3: Graphical representation of all paths from 6 nodes towards Key(111) for a 3-bit wide ID-space (the matching prefix is highlighted, and the corrected bit noted along the path), and the resulting FTT

The implementation of a KBT will decide its *tie-breaking* behaviour, meaning which child will be responsible for its 'virtual' parent. There are two ba-

sic approaches: static or dynamic. Choices of dynamic include: based on age, reliability, load, bandwith or based on a changeable query (cf. subsubsection 3.2.1). Static algorithms can only choose the one that is closest to a a deterministic point of the region (eg. left, right, center). A static choice has the advantage of predictability provided you have a rough estimate about the number of nodes. This predictability allows you to make *informed* guesses about the location of internal nodes of your KBT in your ID space. To our knowledge only SOMO (cf. subsubsection 3.2.4) has this property. For **FFT!**s this is slighlty harder to achieve, since this location is going to be different for each key, but still possible (see [25]).

According to our definitions of Overlays KBTs/**FFT!**s are RO and AO at the same time, i.e. "hybrid".

Summarizing, Monitoring Systems have the choice of maintaining only one global KBT, building a FTT for each query or query-type, or building their own aggregation tree that is not relying on a DHT. Furthermore, they have to decide how to handle aggregation using these trees, once they are built. Data has to be passed up somehow for aggregation. The obvious choice of re-evaluating aggregates on every update of any underlying value might not be the best choice, however. If rarely read values are updated frequently a lot of unused traffic results. Thus a solution could be to only re-evaluate on a read. On the other hand, if frequently requested variables are seldomly written, this strategy leads to a lot of overhead again. Therefore some systems, such as SDIMS (cf. subsubsection 3.2.3), adopt the concept of letting the aggregate requester chose these values. Known as *up-k* and *down-j* parameters, these values specify how far 'up' the tree an update should trigger a re-evaluation, and after this, how far 'down' the tree a result will be passed. Most System implicitly use `all` for these values, this is why SDIMS' k and j parameters default to 'all'.

3 Related Systems

3.1 Distributed Databases

Distributed-*DBMSs* route results from systems that generate query responses towards systems that aggregate them. They are designed to closely approximate the semantics of traditional DBMS. The two most closely related systems are summarized in this section.

3.1.1 PIER

The ***P**2P **I**nformation **E**xchange & **R**etrieval* System [18, 19, 34], which is being developed by the DBMS team at UC Berkeley, is a project meant to be a fully-fledged, distributed query execution engine. It is a DHT-based, flat, relational DB. It has mostly been discontinued in favor of PHI [8].

Like commercial DBMS, it utilizes traditional *boxes-and-arrows* (often called `opgraph`) execution graphs. As its RO Pier uses a DHT. It has successfully been implemented on CAN [42], Bamboo [44] and Chord [48]. P2P Information Exchange & Retrieval (PIER) uses a single-threaded architecture much like SEDA [54], is programmed in Java and is currently deployed on PlanetLab [33, 35] (now called PHI). The basic underlying transport protocol PIER utilizes is Universal Datagram Protocol (UDP), although enriched by the UdpCC Library [44], which provides acknowledgments and Transmission Control Protocol (TCP)-style congestion handling.

A query explicitly specifies an opgraph using their language called UFL. Structured Query Language (SQL) statements that are entered get rendered into an opgraph by the system. This graph is then disseminated to all participating nodes using an index (see below) and is executed by them. For execution, PIER uses an *l*scan operator, which is able to scan the whole local portion of the DHT for values that match the query. To find nodes that might have data for a query, PIER supports three indices that can be used just like indices of any DBMS: true-predicate, equality-predicate and range-predicate. These predicates represent the AO.

Upon joining the network every PIER node sends a packet containing its own ID towards a well-known root that is a-priori hardcoded into PIER. This serve the purpose of a rendezvous point for building the tree. The next hop that sees this packet drops it and notes this link as a child path. These paths form a parent-child relationship among all the nodes, a tree. This is the true-predicate; it reaches every node.

The DHT key used to represent data stored within PIER is the hash of a composite of the table name and some of the relational attributes and carries a random suffix to separate itself from other tuples in the same table. Thus, every tuple can be found by hashing these distinguishing attributes. This is the equality-predicate.

The range-predicate is set by overlaying a Prefix Hash Tree (PHT [43]) on the DHT. It is still not fully supported by PIER.

Once result tuples for a query are produced, they are sent towards the requester by means of the DHT. The query is executed until a specifiable timeout in

the query fires.

For aggregation queries, PIER uses one FTT per query. Every intermediate hop receives an upcall upon routing and gets a chance of changing the complete message; including its source, sink, and next hop.

PIER uses the Java type system for representing data; its tuples are serialized Java objects. They can be nested, composite or more complex objects. PIER does not care about how they are actually represented, other than through their accessor methods.

PIER is unique in its aggressive uses of the underlying DHT, such as for hashing tuples by their keys, finding joining nodes (PHT and indices), hierarchical aggregation and execution of traditional hashing joins. Although [17] suggests that PIER's performance is not as expected, it remains an interesting system for querying arbitrary data that is spread-out over numerous nodes.

3.1.2 Sophia

Sophia [53] is a *Network Information plane*, developed at Princeton and Bekeley. Like other P2P systems, Sophia was evaluated on PlanetLab. A network information plane is a "conceptual" plane that cuts horizontally through the whole network and thus is able to expose all system-state. Consequently, every system described here is an information plane.

While the sophia project does not explicitly address the dynamic structuring of an aggregation operation, its function as a distributed query and aggregation engine is relevant to PlanetenWachHundNetz (PWHN). Sophia does not qualify as a P2P system, from what can be told from [53]. I decided to include it here because it is a system that takes a completely different approach to expressing queries.

Sophia supports three functions: aggregation, distributed queries and triggers. Queries and triggers can be formulated using a high-level, Prolog-based logic language or using the (underlying) low-level, functor-based instruction set. The authors chose a subset of the Prolog-language because both a domain-tailored description and a declarative query language incorporate a-priori assumptions about the system.

Every statement has an implicit part that evaluates to the current NodeID and Time. Thanks to this, Sophia has the ability to formulate statements involving time, as well as caching results. Queries can explicitly state on which nodes to evaluate particular computations. Sophia's query unification engine will expand a set of high-level rules into lower-level explicit evaluations that carry explicit locations. This also gives rise to the ability to rewrite those rules on the fly, thereby allowing in-query re-optimization. In

addition, Sophia's implementation of Prolog has the ability to evaluate and return partial results from "incomplete" statements, i.e. expressions in which some subexpressions cannot be evaluated. An example of this being due to non-reachable nodes.

The run-time core is extendable through loadable modules that contain additional rules. Security within Sophia is enforced through *capabilities*. Capabilities are just aliases for rules that start with the string `cap` and end with a random 128-bit string. To be able to use a particular rule, a user must know its alias because the run-time system makes sure its original name cannot be used. The system also prevents enumerating all defined aliases. To avoid caching aliases in place of the real results, caching is implemented in a way that all capability-references are resolved before storing an entry.

The current implementation on PlanetLab runs a minimal local core on each node, with most of the functionality implemented in loadable modules. All terms are stored in a single, flat logic-term DB. Sensors are accessed through interfaces that insert "virtual" ground terms into the terms-DB, thereby making sensor-readings unifiable (processable by queries).

3.1.3 Summary

D-DBMS are a good approach to expressing arbitrary queries on a distributed set of nodes. Commonly they focus on keeping as much of the ACID semantics as possible intact, while still allowing the execution of distributed queries on a network of nodes. For all intents and purposes of a monitoring system, keeping the ACID semantics is not as important as having greater expressiveness for queries. This is not given in either PIER or Sophia because the former uses SQL whereas the latter uses Prolog to express queries.

3.2 Aggregation Overlays

Aggregation overlays are designed to provide a general framework for collection, transmission and aggregation of arbitrary data from distributed nodes. Generally, they are constructed to allow data that belongs to a specific group, like the group of relational data. This section introduces four aggregation systems.

3.2.1 Astrolabe

Astrolabe [4, 51] is a zoned, hierarchical, relational DB. It does not use an explicit RO as in a DHT. Instead, it relies on the administrator setting up reasonably structured zones according to the topology of the network by assigning proper Domain Name System (DNS) names to nodes. For example, all computers within University of Texas at El Pasos (UTEPs) Computer Science (C.S.) department might start with

"utep.cs." Following the protocol they will all be within the zone named "utep.cs," providing locality advantages over a KBT/FTT approach. Bootstrapping works by joining a hardcoded multicast group on the Local Area Network (LAN). In case that does not work Astrolabe will send out periodic broadcast messages to the LAN, for other nodes to pick up.

Locally available (read: published) data is held in Management Information Bases (MIBs), a name that is borrowed from the Simple Network Management Protocol (SNMP) [7]. The protocol keeps track of all the nodes in its own zone and of a set of *contact nodes* in other zones which are elected by those zones. Astrolabe uses the *gossip protocol* introduced in the introduction.

Each node periodically runs the gossip protocol. It will first update all its MIBs and then select some nodes of its own zone at random. If it is a representative for a zone, it will also gossip on behalf of that zone. To that end it selects another zone to gossip with and picks a random node from its contact list. If the node it picked is in its own zone it will tell that node what it knows about MIBs in their own zone. If the node is in another zone it will converse about MIBs in all their common ancestor zones. These messages do not contain the data; they only contain timestamps to give the receiver a chance to check their own MIBs for stale data. They will send a message back asking for the actual data.

Aggregation functions, introduced in the form of signed certificates, are used to compute aggregates for non-leaf zones. They can be introduced at runtime and are gossiped around just like everything else. Certificates can also be sent that change the behavior of the system. After updating any data, including mere local updates, Astrolabe will recompute any aggregates for which the data has changed.

Summarizing, this means that Astrolabe does not use routing at all. All the information one might want to query about has to be "gossiped" to the port-of-entry node. If one wants to ask a question, one has to install the query and wait until its certificate has disseminated down to all nodes (into all zones) and the answer has gossiped back up to him or her. This makes its *Gossip Protocol* the AO. Thus, Astrolabe does not have an RO according to my definition.

Astrolabe incorporates security by allowing each zone to have its own set of policies. They are introduced by certificates that are issued and signed by an administrator for that zone. For that purpose each zone contains a Certification Authority, which every node in that zone has to know and trust. PublicKey cryptography, symmetric cryptography, and no cryptography at all can all be used in a zone. As-

trolabe at present only concerns itself with integrity and read/write Access Control Lists (ACLs), not with secrecy.

Astrolabe is an interesting project which has some compelling features, like security, but has been superceeded by Willow (cf. subsubsection 3.2.2).

3.2.2 Willow

Willow [52] is a DHT-based, aggregating overlay-tree (a KBT). It uses its own Kademlia-like [28] DHT implementation. It seems to be an implicit successor to Astrolabe since it, according to the authors, inherits a lot of functionality from Astrolabe, at the same time choosing a more well-treaded path. Willow uses TCP and is currently implemented in roughly $2.3k$ lines of Java code (excluding SQL code).

Willow defines its *domains*, much like the zones in Astrolabe, to be comprised of all the nodes that share the same prefix, i.e. that are in the same subtree of the KBT. Following this definition, every node owns its own domain, while its parent domain consists of a node and its sibling.

Like the zones in Astrolabe, every domain elects both a candidate and a contact for itself. The contact is the younger of the two child nodes (or child contacts), whereas the candidate is the older. Thus Willow uses a dynamic election scheme based on age. The contact of a domain is responsible for letting its sibling know about any updates which will then disseminate them down its own subtree.

Willow comes equipped with a tree-healing mechanism but did not inherit Astrolabe's security features.

3.2.3 SDIMS

The ***S**calable **D**istributed **I**nformation **M**anagement **S**ystem* [55, 56] is a system based on FTTs, which is developed at the University of Texas at Austin. It is implemented in Java using the FreePastry framework [45] and has been evaluated on a number of departmental machines as well as on 69 PlanetLab nodes.

The authors state that they designed it to be a basic building block for a broad range of large-scale, distributed applications. Thus, Scalable Distributed Information Management System (SDIMS) is meant to provide a "distributed operating system backbone" to aid the deployment of new services in a network. Flexibility in the context of SDIMS means that it does not assume anything about properties of your data a-priori, for example the update-rate of variables. Instead, up-k and down-j parameters are given while registering a query.

SDIMS respects administrative domains for security purposes by using what is called an Autonomous DHT (ADHT). It does so by introducing superfluous internal nodes into the FTT whenever the isolation of a domain would otherwise have been violated.

Running a query is driven by three functions; install, update and probe.

- *Install* registers an aggregate function with the system. It has three optional attributes: Up, down, and domain.
- *Update* creates a new tuple.
- *Probe* delivers the value of an attribute to the application. It takes four optional arguments: Mode $\in [continuous, one-shot]$, level, up, and down.

SDIMS does not, however, split the aggregation function further into three smaller operators, like in Tiny AGgregation service (TAG).

The current prototype neither implements any security features nor restricts resource usage of a query. Future implementations are planned to incorporate both.

3.2.4 SOMO

The ***S**elf-**O**rganized **M**etadata **O**verlay* [57], which is developed in China, is a system-metadata and communication infrastructure to be used as a *health monitor*. A health monitor can be used to monitor the "health" of a distributed system, meaning it provides information about the system's state (good or bad).

It relies on a so-called *Data Overlay* that allows overlaying arbitrary data on top of any DHT. It facilitates hosting of any kind of data structure on a DHT by simply translating the native pointers into DHT keys (which are basically just pointers to other nodes). To work on a data overlay a data structure has to be translated in the following way:

1. Each object must have a key, and
2. for any pointer A store the corresponding key instead.

Self-Organized Metadata Overlay (SOMO) also stores a last known host along with the key to serve as a routing shortcut. This data overlay on a fairly static P2P network has the ability to give applications the illusion of almost infinite storage space.

SOMO builds this structure, which is very similar to a KBT, on top of an already existing DHT in the following way. The ID space is divided into N equal parts, with $N = Num(Nodes)$. A SOMO node is responsible for one of these parts. From this range a key will be derived in a deterministic way. The default algorithm in SOMO is to take its center. The SOMO node will then be hosted by the DHT node which owns this key.

The SOMO KBT starts out with only one DHT node which will host a SOMO node responsible for the whole ID-space. As soon as a function periodically executed by each SOMO node detects that the ID

range, for which its hosting DHT node is responsible, is smaller than the range it feels responsible for, it will assume that new DHT nodes have joined and spawn new SOMO nodes for them. Thus the tree will grow. The root node will remain responsible for the whole space on level 0 but in lower levels there might be other DHT nodes responsible for certain parts, depending on their location in the ID-space. This scheme does almost, but not exactly, resemble my description of a tie-breaking scheme, with a static algorithm set to the center of the range.

This approach has the disadvantage that it might not react to membership changes as fast as a normal KBT algorithm would.

To gather system metadata each SOMO node will periodically collect reports from its children. If it is a leaf it will simply request the data from its hosting DHT node. Once the aggregate arrives at the root, it is trickled down towards the leafs again.

SOMO can be used for a variety of purposes. It is interesting in its novel approach of layering another abstraction on top of a DHT, which allows it to be somewhat independent of the DHT and have a much simpler design.

3.2.5 Summary

Aggregation overlays are the closest ancestors to what I set out to do. Astrolabe is more a D-DBMS than an aggregation overlay but allows the easy installation of aggregation functions which is why I put it in this section. It uses a gossip-style protocol which is the oldest and most ineffective approach by modern standards.

The available information about Willow is too sparse to be able to even tell if it could be used as a monitoring system, and even less so, how. SDIMS seems to be suitable for this purpose, but still assumes certain attributes about the data that an application needs to aggregate, for example that it has a primitive type and can be aggregated by a function from a small set of prefix functions. SOMO is a much more broader toolkit that can be used to store basically anything in a DHT that can be expressed by reference-types.

3.3 Sensor Networks

Sensor Networks are designed to provide a "surveilance network" built out of commonly small, unmanaged, radio-connected nodes. The nodes have to be able to adapt to an unknown (possibly hostile) territory, organize themselves in a fashion that allows routing and answer queries about or monitor the state of their environment and trigger an action on the occurrence of a specific condition. This section introduces the two most

complete research projects in this field.

3.3.1 IRISNet

The ***I**nternet-scale **R**esource-**I**ntensive **S**ensor **Net**work Services* [30] project at Intel Research is one of the many projects under the hood of IRIS [40]. It is closer to a Sensor Network than the other systems in this survey, which are more comparable to traditional monitoring systems. Prior research of Sensor Networks [5, 22, 27, 39] has primarily focused on the severe resource constraints such systems traditionally faced. IrisNet broadens the definition to include richer sensors, such as internet-connected, powerful, commodity PCs. It provides software infrastructure for deploying and maintaining very large (possibly-planetary-scale) Sensor Networks adhering to this new definition.

IrisNet is a 2-tier architecture. It decouples the agents that access the actual sensor from a database for those readings. Agents that export a generic interface for accessing sensors are called Sensing Agents (SAs). Nodes that make up the distributed database that stores the service specific data are called Organizing Agents (OAs). Each OA only participates in a single sensing service. However, a single machine can run multiple OAs. IrisNet's authors chose eXtensible Markup Language (XML) for representing data because it has the advantage of self-describing tags, thus carrying the necessary metadata around in every tuple. Queries are represented in XQuery because it is the most widely adopted query language for XML data.

Some definitions of P2P demand that a system be called P2P only if it has an address-scheme independent from DNS. According to this definition, IrisNet (as well as Astrolabe) is not P2P since its routing scheme relies on DNS. It names its nodes according to their physical location in terms of the real world. Each OA registers the names of all the SAs that it is responsible for with DNS. Thus it achieves a certain level of flexibility because node ownership remapping is easy, but, on the other hand, is dependent on a working DNS subsystem.

Routing queries to the least common ancestor OA of the queried data is not hard because that name is findable by just looking in the XML hierarchy. The OA that owns this part of the name space can then be found by a simple DNS lookup. If that OA cannot answer all parts of the query, it might send subqueries to other OAs lower in the XML hierarchy. For parts they can answer, OAs also use partially matching cached results. If this is unwanted, a query can specify freshness constraints.

IrisNet lets services upload and execute pieces of code that filter sensor readings dynamically, directly

to the SAs. This is called a *senselet*. Processed sensor readings are sent by the SA to any nearby OA, which will route it to the OA that actually owns this SA. By decoupling the SA from the OA, "mobile" sensors are made possible.

IrisNet's contributions lie more in the field of Sensor Networks, but there is an application of IrisNet to System Monitoring. *IrisLog* runs an SA on each PlanetLab node which uses Ganglia Sensors to collect 30 different performance metrics. Users can issue queries for particular metrics or fully-fledged XPath queries using a web-based form. The IrisLog XML schema describes which metrics should be gathered and to which OA they have to be sent. This is why IrisNet is included here.

3.3.2 TAG

The ***T****iny* ***AG****gregation service* for ad-hoc sensor networks [27], developed at UC Berkeley, is an aggregation system for data from small wireless sensors. It draws a lot of inspiration from Cougar [5] which argues towards sensor database systems. These so called *motes*, also developed at UC Berkeley, come equipped with a radio, a CPU, some memory, a small battery pack and a set of sensors. Their Operating System (OS), called TinyOS, provides a set of primitives to essentially build an ad-hoc P2P network for locating sensors and routing data. The mote wireless networks have some very specific properties that distinguish it from other systems in this survey. A radio network is a broadcast medium, meaning that every mote in range sees a message. Consequently messages destined for nodes not in range have to be relayed.

TAG builds a routing tree through flooding: The root node wishing to build an aggregation tree broadcasts a message with the level set to 0. Each node that has not seen this message before notes the sender ID as its parent, changes the level to 1 and rebroadcasts it. These trees are kept in soft-state, thus the root has to re-broadcast the building message every so often if it wishes to keep the tree alive.

The sensor DB in TAG can be thought of as a single, relational, append-only DB like the one used by Cougar [5]. Queries are formulated using SQL, enriched by one more keyword, `EPOCH`. The parameter `DURATION` to the keyword EPOCH, the only one that is supported so far, specifies the time (in seconds) a mote has to wait before aggregating and transmitting each successive sample.

Stream semantics differ from normal relational semantics in the fact that they produce a stream of values instead of a single aggregate. A tuple in this semantic consists of a $< group\_id, val >$ pair per group. Each group is timestamped and all the values

used to compute the aggregate satisfy $timestamp < time\_of\_sample < timestamp + DURATION$.

Aggregation in TAG works similarly to the other P2P systems that use trees in this survey: The query is disseminated down the tree in a *distribution* phase and then aggregated up the tree in a *collection* phase. The query has a specific timeout in the form of the EPOCH keyword. Consequently, the root has to produce an answer before the next epoch begins. It tells its children when it expects an answer and powers down for the remaining time. The direct children will then subdivide this time range and tell their children to answer before the end of their timeout, respectively.

For computing the aggregates internally that are specified externally using SQL, TAG makes use of techniques that are well-known from shared-nothing parallel query processing environments [47]. These environments also require the coordination of a large number of independent nodes to calculate aggregates. They work by decomposing an aggregate function into three smaller ones:

- an initializer i,
- a merging function f, and
- an evaluator e.

i run on each node will emit a multi-valued (i.e. a vector) *partial-state record* $\langle x \rangle$. f is applied to two distinct partial-state records and has the general structure $\langle z \rangle = f(\langle x \rangle, \langle y \rangle)$, where $\langle x \rangle$ and $\langle y \rangle$ are two partial state records from different nodes. Finally, e is run on the last partial-state record output from f if it needs to be post-processed to produce an answer.

Another unique contribution of this paper is the classification of aggregate functions by four dimensions:

1. *Duplicate sensitive vs. insensitive* Describes a functions' robustness against duplicate readings from one sensor.
2. *Exemplary vs. summary* Exemplary functions return some representative subset of all readings, whereas summary functions perform some operation on all values.
3. *Monotonic* - Describes an aggregate function whose result is either smaller or bigger than both its inputs.
4. *State* - Assesses how much data an intermediate tuple has to contain in order to be evaluated correctly.

What makes this project interesting is not its description of TAG, because I think that its tree building and routing primitives are inferior to others presented in this survey, but its groundwork that is important

in many ways to aggregation systems. The addition of an EPOCH concept to SQL is one such example, while the classification of aggregation functions along four axis is another.

3.3.3 Summary

Whereas both aggregation overlays and distributed DBMS can be used alongside an application that is to be monitored, sensor networks are designed to run by themselves and commonly focus on heavily resource-restricted hardware platforms. While IrisNet is an exception to this, it has no extension mechanism and can only be used to collect Ganglia data through IrisLog. The TAG paper is valuable because of its contribution but otherwise not usable as a monitoring framework.

4 Related Work

The ***D**istributed **A**pproximative **S**ystem **I**nformation **S**ervice* (DASIS) [2] is a system developed at the ETH Zürich, which collects aproximate meta-state about a P2P system. This can be used to improve join and leave algorithms, for example. Dasis is a KBT and can run over a variety of P2P systems. It has been implemented on top of Kademlia [28].

Cone [3] is intended for distributed resource discovery by allowing clients to search for a resource that satisfies a particular property (*max* in this case). They argue that this functionality can be provided by a DHT that has been 'augmented' to permit *heap functionality*. Cone builds a standard KBT for this.

Sensornets (of which [5, 22, 27, 39] are a good cross-cut) are similar to the work presented in this survey, because of their need to present a generic, Database-style interface to distributed data sources. As such they have largely concentrated on a very restricted abstraction of a PC, exemplary called *mote* [27] here, which has limited resources (slow CPU, small memory, limited bandwith). Due to this constraints early contributions have mostly been centered on tiny Operating Systems (*TinyOS*) and low-power network controls. Other works have explored using query techniques for streaming data and using proxies to coordinate queries to save sensors' limited resources. TAG (cf. subsubsection 3.3.2), is actually a SensorNet, but has some similarities to distributed monitoring systems. IrisNet (cf. subsubsection 3.3.1) is classified by its authors as a SensorNet, albeit with more powerful Sensors.

Last but not least, distributed monitoring systems owe a great share to prior work in **Distributed DBMS**, for example Mariposa [49] and R* [24], and **Parallel DBMS**, like Volcano [16] and Gamma [13]. Distributed DBMS, however, have mostly been de-

signed to make distribution transparent to users and still guarantee ACID semantics. Parallel DBMS have pioneered the use of hashing techniques for executing operators in parallel.

5 Summary

Five out of eight Systems presented in this paper are currently deployed on PlanetLab . PlanetLab [33, 35] is consortium of mostly educational institutions around the globe donating machines to a common goal. This goal is to be able to test widely distributed services in their natural habitat, i.e. in a network that is comprised of nodes scattered around the world, connected through the Internet. Naturally, the need to monitor these services arises. In addition to the systems presented here there are the following services on PlanetLab , designed to aid with monitoring of nodes and deploying of new experiments: Management Overlay Networks (MON), Ganglia Monitoring System, CoMon (part of CoDeeN), CoTop (also part of CoDeeN), SWORD, Plush, DSMT which uses PsEPR [6, 41], and last but certainly not least, The PlanetLab Application Manager.

MON [23, 29] builds on-demand trees for distributing software updates and run commands. Commands include returning information about the overlay itself, some statistics returned by the CoTop server and a filter operator. The **Ganglia Monitoring System** [15, 46] builds on UDP multicast within a cluster, and at a higher level on building aggregation trees. Ganglia is a much more general tool and thus not tailored to PlanetLab 's slices and nodes. It uses XML. **CoMon** [10, 32], part of Princeton's CoDeeN project [9], collects per-slice and per-node stats and stores them on a central server, using HTTP for routing and HTML tables for data. **CoTop** [11], looks like a normal 'top' but diplays slice information instead. **SWORD** [31, 50] is meant for distributed resource discovery, for example finding a node with the lowest load. It interfaces with the ganglia, cotop, and trumpet sensors on each node and uses XML for queries, but only MIN/MAX are implemented as of now. **Plush** [1, 38] is meant to provide an extensible execution management system for large scale distributed systems, and was designed with PlanetLab in mind. It will take experiment descriptions in XML and contact remote nodes, upload and run the software and then cleanup. **DSMT** [14] is yet another distributed software deployment tool on PlanetLab , which delivers some statistics about a node on which your software is running. **PlAM** [36] is another deployment tool, that also helps with controlling and monitoring the health of an experiment.

Out of these eight only three could be extended to monitor, send and collect a process' output for aggregation. Plush is modularized and can be extended to collect a process' output, aggregate and send it to a client. MON currently only allows the execution of queries regarding node's resources, but could, according to the authors, easily be extended to query arbitrary sensors. Ganglia allows applications to publish its metrics within ganglia by sending them to its multicast port. Those metrics are then gathered and aggregated by ganglia. The types of those metrics must be explicitly defined by the application in order for ganglia to be able to know how to aggregate them. This restricts applications to a certain set of base types.

Out of the systems presented in this paper that are deployed on PlanetLab , only Pier and SDIMS (and maybe Sophia) seem to be suited to do application level monitoring, albeit they are not making it easy. Both Pier and SDIMS were not designed with users of PlanetLab in mind. Pier was designed to be as close to a local relational query engine as a distributed one could be by giving up some of the ACID (atomicity, consistency, independence and durability) constraints. Its main purpose is therefore not aggregation, nor application-level monitoring. SDIMS is meant to be an aggregating overlay for system metadata. As such it can be used for application-level monitoring by explicitly inserting a process' logs into the system and setting up an aggregation function to reduce the amount of data. None of these choices currently concern themselves with security.

6 Proposal

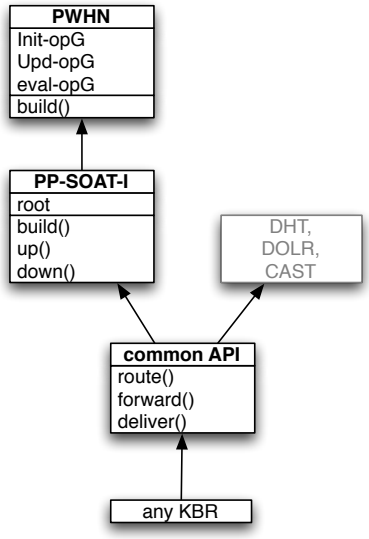

Figure 4: High-level Architecture overview

Researchers use PlanetLab today to set up wide-area distributed experiments, possibly involving hundreds of distinct systems. The software that runs on each node typically needs to be monitored to gather data and track bugs. Currently there is no easy, agreed-upon method of doing that. A researcher can

set up her own system within the application being monitored, but this could potentially alter the readings. Or she could use one of the systems described above, albeit as we already pointed out, this needs some set-up and customization time. Thus, a researcher has a hard time deciding how to do application level monitoring.

We propose to build a software that fills this gap. Implement a software which is meant to facilitate application-level monitoring on PlanetLab. A tool that provides an easy-to-use, always running aggregation abstraction, that researchers could use without the need to do complicated and time consuming setup.

We plan to build a system, called ***P****laneten****W****ach****H****und****N****etz* (PWHN — pronounced '`pawn`'), which is German that loosely translates to 'PlanetaryWatchDogNet', that runs a service on every PlanetLab node and thus does not need to be installed. Researchers would set up an aggregation tree by talking to the local instance of the service and giving it some parameters as well as a security certificate. The interface that this aggregation tree needs to build upon is abstract enough and does not assume anything about the underlying routing strategy so that it can be build on almost anything, including DHTs, Astrolabe's gossiping Algorithm, publish/subscribe and group management protocols. We plan to build a reference module that builds a FTT on the FreePastry Framework which exports the common API for structured Overlays [12]. This has the important advantage that as more and DHTs support this API the choices of Overlays to run PWHN on increase.

As mentioned in the Concepts Section (section 2) a FTT is mostlikely not going to be balanced nor binary. A *Perfect FFT*, meaning one with the exact number of possible nodes, however, could be binary and balanced. A DHTs routing ability depends on the flexibility to choose any node with a given matching prefix while routing. Thus the number of candidates to route to is inversely proportional to the size of the prefix, and vice versa is directly proportional to the suffix. Because of this growing number of choices for the next hop (which also depend on a nodes' fingertable), even a Perfect FTT is most likely not unique, and thus neither binary nor balanced, for any given DHT Key. One possible solution is to restrict a nodes' choice for the next hop to exactly one by also fixing the suffix. For example in a 'normal' DHT a node with $ID = 000$ that wants to route a message to Key(111) has four choices for the next hop: all those that match $1XX$. Fixing the pre- and suffix to the current nodes' ones, while still flipping

the next bit (i.e. adding one more prefix bit) essentially reduces the number of routing candidates to one. Continuing our example, node(000) will have to route the message to node(100), because that corrects the next bit of the (currently emtpy) prefix and keeps the same post-fix. This ensures that the resulting tree will be both binary and balanced for a fully populated ID-space. More likely it will not be fully populated. Rather than (unsuccessfully) trying to route to a large number of non-existing nodes, an application building such a tree should have access to a DHTs fingertable and Key-Based Routing layer (KBR) Layer. However, just adding a single function to the DHTs interface allows an application to build a **PP-FFT!** (**PP-FFT!**) on it. Since the application needs to find nodes without actually storing data at them, which would be the (unwanted) side-effect of a put, the function that needs to be added takes a key and returns the closest node to that key. It could be called FindNode. This data-structure, which we call **PP-FFT!** (**PP-FFT!**), is described in more detail in [25].

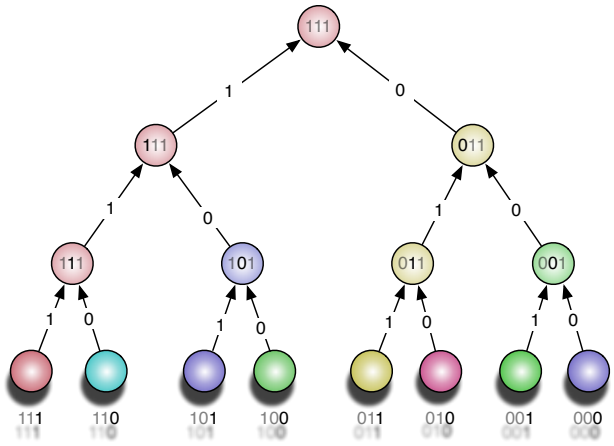

Figure 5: Example of a **PP-FFT!** (the flipped bit is highlighted at each level) for a 3-bit wide ID-Space, all nodes present, for Key(111). For homogeneity of representation the root node is always the left child, thus the right child always has its corresponding bit flipped from the root.

Application of this data structure to aggregation demands two related, albeit orthogonal operations: dissemination 'down' and aggregation 'up' the tree. In the *dissemination phase* a query is broadcasted down the tree to every participating node. The *aggregation phase* does the opposite: the answer from every node is routed up the tree towards the root while being reduced *en-route*. In a **PP-FFT!** routing 'up' the tree is done by fixing both pre- and suffix and flipping bits in the suffix 'from the rear', i.e. adding bits to the front of the matching suffix. Every node only sends one message but needs to wait for the messages of all its children to arrive. The Dissemination phase works the other way around: It starts at the node with the longest matching prefix and flips bits in the prefix 'from the front', thus doing the opposite as in the aggregation phase: it subtracts bits of the matching suffix. Since there are k unique bits (with $k = len(matching\ prefix)$) in the prefix that can be

flipped, the root node needs to send k messages to other nodes, one for each level. This number might actually be smaller, depending on how many of those nodes exist. When represented as a tree (as done in [25]) this can be made to look like a *left-tree*, by sorting the nodes such that the root of the tree is always the left child and its sibling is exactly one flipped bit away.

Parameters to the build call in PWHN include an optional root node, as well as operator-graphs consisting of op(s) or program(s) for the following cases:

- *Initialize* Op-graph that initializes a tuple. It runs on every leaf of the aggregation tree (i.e. every physical node) and translates logs into tuples.
- *Update* Op-graph that reduces the input. It runs on every internal node and will typically take more than one input tuple and produce exactly one output tuple.
- *Evaluate* Op-graph that runs on the root of the tree. It takes the last aggregated value and produces the final output.

This has the advantage of allowing applications to pass arbitrary data around, by allowing them to specify their own aggregators for it. These might merely be a shell script, a php script, or a script in any scripting language. We plan to implement some reference operators that take XML data and perform commonly used functions, like SUM, MAX, AVG etc. XML has the big advantage of carrying its meta-data in the form of tags around so that there is no need for a global schema storage. The local node will then commence to build the tree and set up local endpoints on each node that the user has specified (or all if desired). These endpoints will start calling the init ops or be ready to accept data from the monitored software, by named pipes or sockets for example, and pass it to the init ops. The system will then take care of executing the programs as specified by the op-graphs, passing data up the tree towards the root and finally executing the evaluator op-graph. Operators will only be executed if they can provide the right credentials, as determined by the certificate that was given to the build call.

Our goal is to make application level monitoring of distributed experiments running on PlanetLab as researcher-friendly as possible. Setting up monitoring using our proposed software will take only two steps:

1. Call or send a build message to our system on the PlanetLab node that is to serve as root, i.e. the researchers primary PlanetLab node, telling it your accessor, aggregator and evaluator programs.

2. Write three small programs (could be shell scripts) for the purposes of returning logs for an application, merging to logs into one, and translating the final aggregate into a meaningful output.

And that's it. Our System will take care of the rest.

7 Acronyms

ACL	Access Control List
ADHT	Autonomous DHT
ADT	Abstract Data Type
AO	Aggregation Overlay
C.S.	Computer Science
DBMS	Database Management System
DHT	Distributed Hash Table (see **??**)
DNS	Domain Name System
FTT	FingerTable-based Tree (see **??**)
KBR	Key-Based Routing layer (see **??**)
KBT	Key-Based Tree (see **??**)
LAN	Local Area Network
MIB	Management Information Base
OA	Organizing Agent
OS	Operating System
P2P	Peer-to-Peer
PIER	P2P Information Exchange & Retrieval
PWHN	PlanetenWachHundNetz
RO	Routing Overlay
SA	Sensing Agent
SDIMS	Scalable Distributed Information Management System
SNMP	Simple Network Management Protocol
SOMO	Self-Organized Metadata Overlay
SQL	Structured Query Language
TAG	Tiny AGgregation service
TCP	Transmission Control Protocol
UDP	Universal Datagram Protocol
UTEP	University of Texas at El Paso
XML	eXtensible Markup Language

References

[1] J. Albrecht, C. Tuttle, A. C. Snoeren, and A. Vahdat. Planetlab application management using plush. *SIGOPS Oper. Syst. Rev.*, 40(1): 33–40, 2006. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/1113361.1113370. 5

[2] K. Albrecht, R. Arnold, M. Gähwiler, and R. Wattenhofer. Aggregating information in peer-to-peer system for improved join and leave. In *4th IEEE International Conference on Peer-to-Peer Computing (P2P)*, Aug. 2004. 4

[3] R. Bhagwan, G. Varghese, and G. Voelker. Cone: Augmenting DHTs to support distributed resource discovery. Technical Report CS2003-0755, University of California, San Diego, June 2003. URL citeseer.ist.psu.edu/bhagwan03cone.html. 4

[4] K. P. Birman, R. van Renesse, and W. Vogels. Navigating in the storm: Using astrolabe for distributed self-configuration, monitoring and adaptation. In *Proc. of the Fifth Annual International Workshop on Active Middleware Services (AMS)*, 2003. 3.2.1

[5] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. *Lecture Notes in Computer Science*, 1987:3–??, 2001. URL citeseer.ist.psu.edu/article/bonnet01towards.html. 3.3.1, 3.3.2, 4

[6] P. Brett, R. Knauerhase, M. Bowman, R. Adams, A. Nataraj, J. Sedayao, and M. Spindel. A shared global event propagation system to enable next generation distributed services. In *Proc. of the 1st Workshop on real, large distributed systems (WORLDS)*, Dec. 2004. 5

[7] J. Case, M. Fedor, M. Schoffstall, and J. Davin. RFC 1157 - simple network management protocol (SNMP), May 1990. URL http://www.faqs.org/rfcs/rfc1157.html. 3.2.1

[8] B. Chun, J. M. Hellerstein, R. Huebsch, P. Maniatis, and T. Roscoe. Design considerations for information planes. In *Proc. of the First Workshop on Real, Large Distributed Systems (WORLDS)*, Dec. 2004. URL citeseer.ist.psu.edu/chun04design.html. 3.1.1

[9] CoDeen - A Content Distribution Network for PlanetLab. http://codeen.cs.princeton.edu/. Valid on 13.9.2006. 5

[10] CoMon. http://comon.cs.princeton.edu/. Valid on 13.9.2006. 5

[11] CoTop. http://codeen.cs.princeton.edu/cotop/. Valid on 13.9.2006. 5

[12] 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. 6

[13] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.-I. Hsaio, and R. Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, 1990. URL citeseer.ist.psu.edu/dewitt90gamma.html. 4

[14] DSMT. http://www.dsmt.org/. Valid on 13.9.2006. 5

[15] Ganglia. http://ganglia.info/. Valid on 13.9.2006. 5

[16] G. Graefe. Encapsulation of parallelism in the volcano query processing system. In *Proceedings of the ACM SIGMOD international conference on Management of data (SIGMOD)*, pages 102–111, New York, NY, USA, 1990. ACM Press. ISBN 0-89791-365-5. doi: http://doi.acm.org/10.1145/93597.98720. 4

[17] R. Huebsch, B. Chun, and J. Hellerstein. Pier on planetlab: Initial experience and open problems. Technical Report IRB-TR03 -043, Intel Research Berkeley, Nov. 2003. URL citeseer.ist.psu.edu/huebsch03pier.html. 3.1.1

[18] 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. 3.1.1

[19] 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. 3.1.1

[20] JXTA. http://www.jxta.org. Valid on 13.9.2006. 1

[21] M. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, Mar. 2003. URL citeseer.ist.psu.edu/kaashoek03koorde.html. 2

[22] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for 'Smart Dust'. In *Proc. of the*

International Conference on Mobile Computing and Networking (MOBICOM), pages 271–278, 1999. URL citeseer.ist.psu.edu/kahn99next.html. 3.3.1, 4

[23] J. Liang, S. Y. Ko, I. Gupta, and K. Nahrstedt. Mon: On-demand overlays for distributed system management. In *Proc. of the 2nd USENIX Workshop on Real, Large Distributed Systems (WORLDS)*, Dec. 2005. 5

[24] G. M. Lohman, C. Mohan, L. M. Haas, D. Daniels, B. G. Lindsay, P. G. Selinger, and P. F. Wilms. Query processing in R*. In *Query Processing in Database Systems*, pages 31–47. Springer, 1985. 4

[25] 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. 2, 6, 6

[26] V. Lorenz-Meyer and E. Freudenthal. Scalable and locality-aware MapReduce for p2p systems with dynamic membership. Unpublished, Apr. 2006. URL http://rlab.cs.utep.edu/~vitus. Master's Thesis proposal. 1

[27] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/844128.844142. 3.3.1, 3.3.2, 4

[28] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Mar. 2002. URL http://www.cs.rice.edu/Conferences/IPTPS02. 2, 3.2.2, 4

[29] MON. http://cairo.cs.uiuc.edu/mon. Valid on 13.9.2006. 5

[30] S. Nath, Y. Ke, P. B. Gibbons, B. Karp, and S. Seshan. Irisnet: An architecture for enabling sensor-enriched internet service. Technical Report IRP–TR–03–04, Intel Research Pittsburgh, June 2003. 3.3.1

[31] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Distributed resource discovery on planetlab with SWORD. In *Proc. of the 1st Workshop on Real, Large Distributed Systems (WORLDS)*, Dec. 2004. 5

[32] K. Park and V. S. Pai. Comon: a mostly-scalable monitoring system for planetlab. *SIGOPS Oper. Syst. Rev.*, 40(1):65–74, 2006. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/1113361.1113374. 5

[33] 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. 1, 3.1.1, 5

[34] PIER. http://pier.cs.berkeley.edu/. Valid on 13.9.2006. 3.1.1

[35] PlanetLab. http://www.planet-lab.org. Valid on 13.9.2006. 1, 3.1.1, 5

[36] Planetlab Application Manager. http://appmanager.berkeley.intel-research.net/. Valid on 13.9.2006. 5

[37] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997. URL citeseer.ist.psu.edu/plaxton97accessing.html. 2

[38] Plush. http://plush.ucsd.edu/. Valid on 13.9.2006. 5

[39] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Commun. ACM*, 43 (5):51–58, 2000. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/332833.332838. 3.3.1, 4

[40] Project IRIS. http://www.project-iris.net. Valid on 13.9.2006. 1, 3.3.1

[41] PsEPR. http://psepr.org/. Valid on 13.9.2006. 5

[42] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. Technical Report TR-00-010, University of California, Berkeley, Berkeley, CA, 2000. URL citeseer.ist.psu.edu/ratnasamy02scalable.html. 2, 3.1.1

[43] S. Ratnasamy, J. Hellerstein, and S. Shenker. Range queries over DHTs. Technical Report IRB-TR-03-009, Intel Research, 2003. 3.1.1

[44] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proc. of USENIX Technical Conference*, June 2004. URL citeseer.ist.psu.edu/rhea03handling.html. 2, 3.1.1

[45] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–??, 2001. URL citeseer.ist.psu.edu/rowstron01pastry.html. 2, 3.2.3

[46] 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 Conference*, Hong Kong, 2003. 5

[47] A. Shatdal and J. F. Naughton. Adaptive parallel aggregation algorithms. In *Proceedings of the ACM-SIGMOD international conference on Management of data (SIGMOD)*, pages 104–114, May 1995. URL citeseer.ist.psu.edu/shatdal95adaptive.html. 3.3.2

[48] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001. URL citeseer.ist.psu.edu/stoica01chord.html. 2, 3.1.1

[49] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A wide-area distributed database system. *VLDB Journal: Very Large Data Bases*, 5 (1):48–63, 1996. URL citeseer.ist.psu.edu/stonebraker96mariposa.html. 4

[50] SWORD. http://www.swordrd.org/. Valid on 13.9.2006. 5

[51] R. van Renesse and K. P. Birman. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining, 2001. URL citeseer.ist.psu.edu/article/robbert01astrolabe.html. 2, 3.2.1

[52] R. van Renesse and A. Bozdog. Willow: DHT, aggregation, and publish/subscribe in one protocol. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2004. URL citeseer.ist.psu.edu/642279.html. 3.2.2

[53] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An information plane for networked systems, Nov. 2003. URL citeseer.ist.psu.edu/wawrzoniak03sophia.html. 3.1.2

[54] M. D. Welsh. *An Architecture for Highly Concurrent, Well-Conditioned Internet Services*. PhD thesis, University of California, Berkeley, 2002. URL citeseer.ist.psu.edu/532228.html. 3.1.1

[55] P. Yalagandula. *A Scalable Information Management Middleware for Large Distributed Systems*. PhD thesis, University of Texas at Austin, Aug. 2005. 3.2.3

[56] P. Yalagandula and M. Dahlin. A scalable distributed information management system, 2003. URL citeseer.ist.psu.edu/yalagandula03scalable.html. 3.2.3

[57] Z. Zhang, S. Shi, and J. Zhu. Somo: Self-organized metadata overlay for resource management in p2p DHT. In *Proc. of the Second Int. Workshop on Peer-to-Peer Systems (IPTPS)*,

Berkeley, CA, Feb. 2003. URL citeseer.ist.psu.edu/zhang03somo.html. 3.2.4

[58] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Apr. 2001. URL citeseer.ist.psu.edu/zhao01tapestry.html. 2

Table 1: Overview of Related Systems

Name	Type	RO	AO	Deployed on	Language	Notes
PIER	query exec.	DHT	FTT	PlanetLab	UFL	Java Type System
Sophia	query exec.	static	static	PlanetLab	Prolog	
Astrolabe	hier. aggr. DB	DNS	gossip	-	SQL	Security
Willow	query exec.	KBT	KBT (dyn:age)	-	SQL	
SDIMS	hier. aggr. DB	ADHT	FTT	Dept. + PlanetLab	aggreg. : C	
SOMO	meta-data	DHT	KBT (stat:center)	-		host any ADT
IrisNet	SensorNet	DNS	OA	PlanetLab	XPath	"rich" sensors
TAG	query exec. (SN)	broadcast	simple tree	PlanetLab	enriched SQL	*mote*