San Fermín: Aggregating Large Data Sets using Dynamic Binomial Trees*

Justin Cappos and John H. Hartman
Department of Computer Science
University of Arizona
Tucson, AZ, 85721
{justin,jhh}@cs.arizona.edu

Abstract

This paper presents San Fermín, a system for aggregating large data sets from the nodes of large–scale distributed systems. Each San Fermín node individually computes the aggregated result by dynamically creating its own binomial tree as it aggregates data. Nodes that fall behind abort their aggregations, thereby reducing overhead. Having each node create its own binomial tree makes San Fermín highly resilient to failures, and ensures that the internal nodes of the tree have high capacity, reducing completion time without overwhelming nodes.

Compared to existing solutions San Fermín handles large data sets better, has higher completeness when nodes fail, computes the aggregated result faster, and has better scalability. We analyze the completion time, completeness, and overhead of San Fermín versus existing solutions using analytical models, simulation, and experimentation with a prototype deployed on PlanetLab. Our evaluation shows that San Fermín is scalable both in the number of nodes and in the size of the data being aggregated. With 10% node failures during aggregation, San Fermín still returns the answer from over 97% of the nodes and in most cases does so faster than the underlying DHT recovers from failures.

1 Introduction

The goal of this research is to aggregate large data sets stored in large–scale distributed systems efficiently. For example, CERT logs about 1/4 TB of information daily on nodes distributed throughout the Internet [2]. Analysts must use these logs to detect the anomalous behavior that signals worms and other attacks, and must do so quickly to minimize damage. An example query might request the number of flows to and from each TCP/UDP port (to detect an anomalous distribution of traffic indicating an attack). The challenge is to provide high *completeness* (fraction of nodes whose data are included in the result) and low *completion time*, while tolerating node failures and without overwhelming nodes.

Current aggregation systems are designed for small amounts of aggregate data (typically only a few bytes) [4, 20, 28]; they make use of techniques such as balanced trees and continuous queries over trees with high–degree internal nodes. High–degree internal nodes that are concurrently used reduce latency when the amount of data is small, but can be overwhelmed when the amount of data is large. This slows the aggregation and increases the peak network traffic observed by a node, which increases the potential for failures to reduce completeness.

In this paper we present San Fermín[1], an aggregation technique based on binomial trees that is efficient, dynamic, and scalable. Compared to trees with high–degree internal nodes, dynamic binomial trees offer higher completeness, faster results, and improved resilience to failure for large aggregate data sizes. San Fermín leverages DHT (Distributed Hash Table) technology to allow each node to construct its own binomial aggregation tree by successively partnering with other nodes increasingly distant in node ID space. Nodes race to complete the aggregation while nodes that fall behind abort their aggregations. This naturally ensures that high–capacity nodes perform the bulk of the aggregation, while limiting the effect of node failures and slow nodes. Having each node compute the aggregation allows San Fermín to tolerate many failures, since a node that does not fail will compute an aggregation of some subset of the nodes, perhaps only itself.

1.1 Contributions

San Fermín makes use of dynamic binomial trees whose performance scales well with the data size and number of nodes, and are highly–tolerant of failures. By limiting the number of nodes contending for any node's

*This work was supported in part by the NSF under grant CCR–0435292

[1] San Fermín is the festival at Pamplona that includes the running of the bulls.

resources, San Fermín significantly decreases per–node network traffic, while providing high completeness and low completion time.

Analytic models demonstrate that San Fermín scales better in both network size and aggregate data size than centralized, balanced tree, and supernode solutions. Prototype implementations of San Fermín and SDIMS [28] are compared on PlanetLab to demonstrate the differences on a real world network. Simulation results also show that San Fermín scales well, has low overhead, and provides high completeness even in the presence of failures.

1.2 Applications

In addition to aggregating network information as in the CERT example, San Fermín also benefits other applications that require aggregating large amounts of data from many nodes:

Software Debugging Recent work on software debugging [17] leverages execution counts for individual instructions. This work shows how the total of all the instruction execution counts across multiple nodes helps the developer quickly identify bugs.

System Monitoring Administrators often wish to process the logs of thousands of nodes around the world to troubleshoot difficulties, track intrusions, or monitor performance.

Distributed Databases A common operation in relational databases is a GROUP BY query [22]. This query combines table rows containing the same attribute value using an aggregate operator (such as SUM). The query result contains one table row per unique attribute value. In distributed databases different nodes may store rows with the same attribute value and this information must be combined and returned to the requester.

These applications are characterized by their need for the aggregate result of a large amount of data. In most cases the aggregate data from multiple nodes can be aggregated to produce a result that is approximately the same size as any individual node's data. The target environments may contain hundreds or thousands of nodes, requiring the aggregation to tolerate failures.

1.3 Limitations

San Fermín focuses on *one-shot* queries rather than continuous queries. In a continuous query, information is continually streamed from the leaves to the root. This reduces the requester's latency in receiving updated aggregate data at the cost of increasing overhead. As the size of the aggregated data increases, the overhead quickly becomes excessive because each update sends aggregate data up to the root. Due to this scalability and efficiency limitation, San Fermín does not support continuous queries.

Aggregations in San Fermín are only performed on nodes that were alive when the request was issued. Nodes that come up during an aggregation will not be included in the result.

2 San Fermín Overview

Each node in San Fermín performs its own aggregation by creating an individual binomial tree based on nodeIds in the underlying DHT. Each node initially partners with another node that has the longest nodeId prefix in common. These nodes exchange their data and each compute the aggregated result. Each then moves on to partner with a node that has the second-longest nodeId prefix in common, and so on until it partners with a node that has no nodeId prefix in common. At this point the node has computed the aggregated result; the first node to do so sends it to the requester, and the rest of the nodes halt their aggregations. To reduce overhead and improve completeness, nodes that fall behind during the aggregation process abort their aggregations. San Fermín is highly resilient to failures since all nodes start partnering with other nodes when the aggregation begins; once a node has partnered with another node both nodes must fail for its data to be lost, and as the aggregations progress the number of nodes that must fail grows exponentially. San Fermín uses a timeout mechanism to detect node failures, but scales node polling intervals based on nodeId prefixes to ensure nodes are not overwhelmed. Details of the San Fermín implementation are presented in the next section.

We had several design goals for San Fermín:

Completeness San Fermín maximizes the number of nodes included in the result, even if nodes fail. Completeness can be less than one due to node failures and in general should be no worse than the number of nodes that fail (barring catastrophic events like partitioning the DHT). An aggregation system can do better than that depending on how it handles node failures.

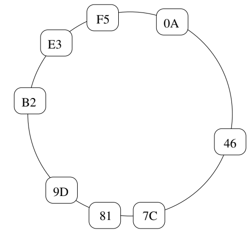
Speed San Fermín minimizes completion time to improve the completeness and responsiveness of the system.

Correctness San Fermín should produce *correct* results, so that the data from a node should appear exactly once in the result. In the terminology used by Jain et al. [13], San Fermín has perfect arithmetic precision in the returned results, temporal imprecision only due to clock skew amongst nodes, and nearly perfect network imprecision in the face of failures without the possibility of duplicated results.

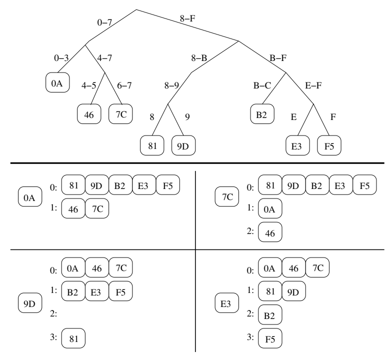
Dynamic Trees The nodes and the network components have inherent differences in capacities, and these capacities can vary over time due to loads and contention for resources. Efficiently constructing and maintaining a static tree in a dynamic environment is challenging. San

Fermín instead constructs dynamic trees as the aggregation progresses. Nodes that fall behind in constructing their trees abort the process, naturally ensuring that low–capacity or overloaded nodes do not slow down the aggregation process.

Data Exchanges When nodes in San Fermín partner they exchange their data, rather than simply have one send its data to the other. This allows each to perform its own aggregation and create its own binomial tree, leading to high completeness and low completion time. The downside is that the nodes transfer twice as much data as strictly necessary, as at most one of the nodes in a partnering will eventually produce the final aggregated result. There are two reasons why the additional network traffic is acceptable. First, the bandwidth bottlenecks tend to be at the edge of the network [12]. This means that a node can saturate its bottleneck without affecting other nodes. Second, most edge networks are switched and the links are full–duplex (with the notable exception of wireless); this means that the node can send and receive data simultaneously without reducing the bandwidth available to either. Thus the cost of the increased network traffic is more than offset by the increased completeness and improved completion time it affords.



(a) Pastry Ring



(b) Sample Prefix Tables

Figure 1: Pastry Ring and Prefix Tables

3 San Fermín Details

This section describes the details of San Fermín, including an overview of the Pastry DHT upon which the prototype in layered, a description of how San Fermín nodes find other nodes with whom to exchange, how failures are handled, how timeouts are chosen, and how laggards are aborted.

3.1 Pastry

Pastry [23] is a peer-to-peer based DHT abstraction similar to Chord [25] and Tapestry [32]. Each node has a unique 160-bit nodeId that is used to route messages and identify nodes. Given a message and a destination nodeId Pastry routes the message to the node whose nodeId is numerically closest to the destination nodeId.

Each Pastry node has two structures used to route messages: a *routing table* and a *leaf set*. The leaf set consists of a fixed number of nodes that have the numerically closest nodeIds to the current node. This assists nodes in the last step of routing messages and in rebuilding routing tables when nodes fail.

The routing table consists of node information organized in rows by the length of the common prefix. When routing a message in the nodeId space, a node forwards the message to the node in the routing table with the longest prefix in common with the destination nodeId.

Pastry uses nodes with nearby network proximity when routing tables are constructed. As a result, it has been shown that the average latency of Pastry messages is less than twice the IP delay [5]. For a complete description of Pastry see the paper by Rowstron and Druschel [23].

3.2 Finding Partners

Each San Fermín node computes the aggregated result of all nodes by exchanging data with other nodes having aggregate data from progressively larger and larger sets of nodes. The exchanges are based on nodeId prefixes: initially each node exchanges data with a node that has the longest nodeId prefix in common, then progresses through a series of exchanges with nodes that have shorter and shorter prefixes in common, and finally exchanges data with a node that has no prefix in common, thereby getting the aggregated data from the other half of the nodeId space. For example, in Figure 1 there are 8 nodes aggregating data. Nodes 46 and 7C will initially exchange data because they have the longest matching prefix with each other (as will 81 and 9D, and E3 and F5). One of 46 and 7C will then exchange with 0A, and one of those will finally exchange with a node from the other subtree whose nodeId starts with 8–F. Barring a failure, at this point the node's data are the result of aggregating the data from every node. Having every node perform the aggregation enables San Fermín to tolerate failures and reduces the completion time. Once one node produces the result the remaining aggregations are aborted on all other nodes. As San Fermín runs it continually prunes aggregations that are unable to find partners with whom to exchange data.

The aggregation process is driven by a *prefix table* maintained on each node. There is one row in the table for each prefix of the node's nodeId, and each row contains a list of known nodes with that particular prefix (if any). When an aggregation begins the prefix table is initialized from Pastry's leaf set and routing table. Pastry fills its routing table with nodes having favorable network characteristics; this biases each San Fermín node to exchange data with nodes with which it has high–performance network connections, improving aggregation performance. Pastry's routing table and leaf set are fixed-size, so the prefix table is also fixed-size.

The data exchange process has three steps: finding a partner; exchanging data; and aggregating data. A node uses *invitations* to invite other nodes to exchange data for a particular prefix. Whenever a node is ready to exchange data for a prefix it first attempts to do so with all nodes from whom it has received invitations for that prefix. If an exchange succeeds the node moves on to exchange data on the next shorter prefix. If the exchanges fail with all nodes that sent invitations, the node sends its own invitations to all nodes in the corresponding row of its prefix table and begins a series of timeouts waiting for a response, sending invitations to larger and larger sets of nodes after each timeout before eventually declaring the prefix dead and moving on to the next shorter prefix. In Figure 1, node 0A sends invitations to its longest prefix matches 46 and 7C and then begins its series of timeouts. After 46 and 7C finish exchanging data they each respond to those invitations by attempting to exchange with 0A. Whichever is unsuccessful in exchanging with 0A will instead instead send invitations to the prefix and wait for a timeout.

Exchanging data with another node consists of establishing a connection with that node and verifying that both nodes agree about the prefix to be aggregated. If not, the connection is dropped and the nodes resume looking for a partner. If the prefixes do match, the nodes refuse all other exchange requests for the current prefix and exchange their data. Once they have exchanged the data they each to compute the aggregation of the two data sets and move on to the next shorter prefix. If there is no shorter prefix to aggregate then they have the complete answer for the entire tree and instead provide the requester with the answer.

A node performs the following actions when it receives an invitation. If the node's current prefix (the prefix for which it is currently trying to find a partner) is shorter than the invited prefix then the node has already exchanged data for the invited prefix, so the node replies with "No". Otherwise, the node may be willing to exchange data for the invited prefix in the future so it replies with "Maybe"and adds the sender to a *ready table* that keeps track of potential future partners. If the invited prefix matches the node's current prefix and the node has not yet started to exchange data, the node connects to the sender directly. If the sender has already aggregated the invited prefix the exchange will fail. In Figure 1, when 0A sends invitations to 46 and 7C they each respond with a Maybe to signify they are alive and have not exchanged data with that prefix yet. 0A will continue to send periodic invitations until either 46 or 7C exchanges data with it. Suppose 7C initiates the exchange with 0A. 0A will respond with Maybe to any invitations from 46 (and should 7C fail during the exchange 0A will exchange with 46). Once the exchange from 0A and 7C completes, 0A will respond to any further invitations from 46 with a No.

If a node is unable to exchange data with nodes from whom it received invitations it actively seeks out a partner. First, it sends invitations to all nodes in the corresponding row of the prefix table, sets a timeout, and waits for a reply. If it receives a No it simply makes note of it and continues to wait. If it receives a Maybe it knows that sender received the invitation and is a possible future partner so the node resets the timeout and resumes waiting. If at any time a connection is established and an exchange is successful the node ceases to wait and simply moves on to the next shorter prefix.

If the timeout expires without receiving a reply the node resends an invitation to the last node that responded with a Maybe to a previous invitation. If a Maybe is not received to the second invitation the node assumes that there is a problem with the node that originally sent the reply, so the node resends an invitation to that node as well as to a random node in the row and a random nodeId with the proper prefix. If a Maybe is not received for these invitations the node sends invitations to the last node to reply, all nodes in the row, and $n + 1$ random nodeIds with the proper prefix, where n is the number of nodes in the row. If a Maybe is not received to these invitations the node looks to see if a No was received at any point during the invitation process. If so, there is another node that has already aggregated the prefix so the current node simply aborts. If a No was not received the node declares the prefix dead and moves on to the next shorter prefix.

At first glance sending invitations to random nodeIds appears pointless — sending them to the nodes in the routing table seems sufficient because these nodes are known to exist and Pastry will eventually notify San Fermín if they fail. Furthermore, messages sent to random nodeIds will be routed through the nodes in the routing table, so if San Fermín cannot communicate with the nodes in the routing table for a prefix then it cannot communicate with any nodes in the prefix. There are two reasons for sending invitations to random nodeIds anyway. First, the nodes in the routing table may have aborted the aggregation (see Section 3.5), but there may be other nodes in the prefix that have not. Second, the nodes in the routing table may not respond to San Fermín invitations, but still route Pastry messages (see Section 3.3). In both of these cases the nodes in the routing table are not potential partners, but can route messages to potential partners.

Empty rows in the prefix table are handled in one of two ways. First, the Pastry leaf set for a node contains the node's k immediate neighbors in nodeId space. If the row falls within the leaf set then the row is empty because no nodes have that prefix. In this case San Fermín skips the row. In Figure 1, this is the case for 81 and 9D who have no node with a 2–bit prefix match. Otherwise, if the row does not fall within the Pastry leaf set then nodes may exist with that prefix, but the current node does not know about them. In this case the node sends out invitations as described above. Note that having an empty row that does not fall in the leaf set is unlikely, since each shorter prefix doubles the range of nodeId space covered.

3.3 Handling Failures

The Pastry layer on a node notifies the San Fermín layer when the routing table or leaf set changes, allowing San Fermín to detect some failures directly, but node failures outside of the routing table or leaf set will not be reported, nor will failures that cause the node to stop responding to San Fermín messages while continuing to respond to Pastry messages. In addition, it may take a significant amount of time for Pastry's routing tables to converge after a failure.

San Fermín is able to tolerate two types of node failures: *dead* nodes and *zombie* nodes. A dead node is one recognized as no longer functioning by Pastry, and is removed from any routing tables and leafs sets in which it appears. A zombie node is considered functional by Pastry, but no longer responds to any San Fermín messages sent to it. San Fermín handles these two types of failures differently.

San Fermín is notified by Pastry of a dead node and changes its prefix table to accordingly. Unless the change affects the current prefix and the node does not have a partner for that prefix, no further action is necessary. Otherwise, if the dead node lies within the leaf set and the prefix is now empty, the node moves on to the next shorter prefix. If a node is removed from Pastry that is not within the leaf set, San Fermín does not skip the prefix even if its routing table entries are empty because Pastry's routing tables take some time to converge.

A zombie node is not detected by Pastry, but is noticed by San Fermín because the node stops responding to invitations. As long as an San Fermín node can contact at least one responsive node in the current prefix, the node will continue to wait on that prefix. This ensures that San Fermín will only timeout on prefixes that have no live nodes. If in Figure 1, nodes 46 and 7C become zombie nodes then 0A will eventually time out their subtree. The 8-F nodes in the system will not timeout the entire 0–7 subtree as long as they can contact 0A.

3.4 Choosing Timeouts

San Fermín relies on timeouts to determine when to resend invitations and ultimately when to declare a prefix dead and move on. The timeout length is important because it should be short enough so that San Fermín resends invitations promptly, but not so short that San Fermín prematurely declares a prefix dead. San Fermín accomplishes this by scaling the timeout for a batch of invitations sent to a prefix based on the number of nodes with that prefix — the more nodes with a prefix the less likely they will all fail, and so the longer the timeout. Longer timeouts are also beneficial because they give time for the Pastry routing tables to converge. If a prefix falls within a node's leaf set the node knows exactly how many nodes have the prefix and sets the timeout to that number times a constant c (2 seconds by default). Otherwise the node must estimate the number of nodes that have the prefix.

The estimate of the number of nodes that have a given prefix is derived from the estimated number of nodes in the entire system, which in turn is estimated from the range of nodeIds spanned by the Pastry leaf set for the node. The hashing function distributes the nodeIds uniformly so the average distance between all n nodeIds is approximately $\frac{1}{n}$ of the nodeId space. This allows the total number of nodes n to be estimated by measuring the average distance d between a subset of nodes (specifically the leaf set) and computing what fraction of the nodeId space it covers. A prefix of b bits covers $\frac{1}{2^{b+1}}$ of the nodeId space, therefore the estimated number of nodes m with a given b-bit prefix is $\frac{n}{2^{b+1}}$. San Fermín then sets the timeout to mc. An important property of this technique is that the timeouts for prefixes that contain fewer nodes are shorter than the prefixes containing more nodes. This causes San Fermín to timeout the longest prefix possible and exclude only unresponsive nodes.

Making the timeout proportional to the number of nodes in a prefix also allows San Fermín to scale without overwhelming nodes with invitations. Consider a node whose current prefix is b bits long. It will respond with Maybe to nodes whose current prefixes are shorter or equal to its own, and No to nodes with longer prefixes. The former will continue to send invitations to the current node. Since nodeIds are uniformly distributed, on average the node will receive one invitation per prefix and there are b prefixes shorter than or equal to its current prefix. The timeout for a prefix of length b is $\frac{cn}{2^{b+1}}$ which corresponds to a rate of $\frac{2^{b+1}}{cn}$. The invitation rate from all prefixes is therefore $\sum_{i=0}^{b} \frac{2^{i+1}}{cn} \leq \sum_{i=0}^{\lceil log_n - 1 \rceil} \frac{2^{i+1}}{cn} = \frac{2(2^{\lceil log_n \rceil}-1)}{cn} \leq \frac{4(n-1)}{cn} < \frac{4}{c}$ which is a constant.

3.5 Aborting Laggards

San Fermín aborts aggregations by nodes that cannot find a partner for a prefix that other nodes have already aggregated. If a node times–out on a prefix without finding a partner it checks to see if it received a No to an invitation. If so, the node aborts, otherwise it considers the prefix dead and moves on. Having nodes abort is a tradeoff between completion time and completeness. If a node were instead to continue, its aggregation would not include nodes from the skipped prefix, reducing its completeness. If, on the other hand, San Fermín were modified so that nodes did not respond with No to an invitation but exchanged instead, then nodes that already aggregated a prefix could be forced to aggregate the prefix multiple times, slowing down the overall aggregation. San Fermín balances these considerations by forcing a node that cannot find a partner to abort. Note that this could affect the overall aggregation if a node aborts because other nodes already aggregated a prefix but all nodes that did so subsequently failed. In this case the aggregated result will not contain any data from the prefix when in theory it could have contained data from the aborted node. For example, in Figure 1 suppose after nodes 7C and 0A fail after they exchange data with each other but before they exchange data with any of the 8–F nodes. If 46 were to abort before 7C and 0A failed then the result would not contain data from 46. In practice this is unlikely to happen because each exchange increases the number of nodes with the data from a prefix, making it more and more unlikely that they will all fail.

Aborting also has the beneficial side–effect of reducing overhead, as discussed in Section 4.3.4. A node that aborts does not continue aggregating data and ceases to incur overhead. If the node was not going to produce the aggregated result first this results in a net reduction in overhead without affecting completeness or completion time.

4 Evaluation

We evaluated San Fermín using three techniques: analytical models; simulations; and experiments using a prototype implementation on PlanetLab. These techniques enable exploration of the tradeoffs between San Fermín and existing techniques in terms of overhead, completion time, and completeness. They also allow analysis of San Fermín behavior such as the resilience to failures and the variance in overhead at different nodes.

4.1 Analytical Models

The analytical models enable comparison of completion time and completeness of four different techniques for aggregating data: centralized; binomial trees (San Fermín); balanced trees (SDIMS[2]), and Supernodes (Seaweed). The models use system parameters measured from real world systems. In the models the total number of nodes is denoted as N. The *churn rate* c is the fraction of the nodes N expected to fail per second. The models assume that any node that fails during the aggregation does not recover, and any node that comes online after the aggregation begins does not join the aggregation. A node that fails while sending data causes the entire send to fail. Inter–node latencies and bandwidths are a uniform l and b, respectively. The bandwidth b is considered a per node limitation, which is consistent with real world observations that the bandwidth bottleneck is usually at the edges of the network and not in the middle [12]. Each node contributes data of size s and the aggregation function condenses all input data to a result of size s. Per-packet, DHT, and connection establishment costs are ignored for all techniques.

4.1.1 Centralized (Direct Retrieval)

In the centralized model, the requester contacts every node and retrieves its data directly. The requester then aggregates all of the data locally. The requester can eliminate almost all latency costs by pipelining the retrievals so that the next one starts as soon as the current completes. The time to retrieve the data from all nodes is therefore:

$$l + \frac{s * N}{b} \tag{1}$$

The result completeness is the number of nodes that did not fail prior to the requester retrieving their data. The probability that a node is alive after t seconds is $(1-c)^t$, so the expected completeness is:

$$\sum_{i=1}^{N} (1-c)^{\frac{i*s}{b}+l} \tag{2}$$

[2]SDIMS nodes use Pastry routing tables to form aggregation trees by using the next hop toward the root key as their parent. Since a single node with good latency properties commonly occurs in many nodes' routing tables, this leads to internal nodes having many children (we have observed similar behavior by Scribe [6] and so believe this property is inherent in the technique and not an artifact of the implementation). As a result we classify SDIMS as building balanced trees. When using a DHT like Chord [25], SDIMS builds binomial trees.

4.1.2 Binomial Trees (San Fermín)

The analytical model assumes a complete binomial tree to simplify analysis, leading to the following completion time:

$$\log_2 N * (\frac{s}{b} + l) \tag{3}$$

The completeness of using binomial trees is superior to the centralized solution because after a node partners with n other nodes its data will appear in 2^n binomial trees, meaning that 2^n nodes must fail for the original node's data to not be included in the result. The probability of single node failing by time t is $1 - (1 - c)^t$, and the probability of a group of g nodes all failing by time t is $(1 - (1 - c)^t)^g$. This expected completeness is therefore:

$$N - \sum_{i=1}^{\log_2 N} \frac{N}{2} * (1 - (1 - c)^{i*(\frac{s}{b}+l)})^{2^{i-1}} \tag{4}$$

4.1.3 Balanced Trees (SDIMS)

Aggregation is often performed using trees in which the internal nodes have a similar degree d and the majority of leaf nodes have similar depth. An internal node waits for all of its child nodes to send it data after which it computes the aggregation of all the child data and its own data and sends the result to its parent. In practice, one of the child nodes is also the parent node so only $d-1$ nodes at the lower level must send data to it.

The model assumes that trees are balanced and complete with degree d. In the case of an internal node failure, a new node is selected to take the place of the failed internal node. All of the child nodes of the failed node must resend their data to their new parent. Since the parent is chosen from the child nodes at the lower level, an internal node failure causes data to be resent from one internal node in each of the lower levels. This may introduce a variable amount of delay in the parent's response (depending on the timing and level of the failures). The model assumes these failures do not affect completion time to simplify analysis. The completion time is:

$$\log_d(N) * (\frac{(d-1)*s}{b} + l) \tag{5}$$

The completeness is affected by node failures. In the common case, a node that fails before sending to its parent will be excluded from the result. It is also possible that both the child and parent fail after the child has sent the data, causing the child to be excluded. The completeness model captures these node failures, but does not consider a cascade effect involving a failure of the failed child's children. Since with $\sum_{i=0}^{\log_d(N)-1} \frac{N}{(d-1)*d^i}$ nodes per level there is a $\sum_{j=1}^{d-1}(1-(1-c)^{j*\frac{s}{b}+l})$ probability of an internal node failure with $\sum_{k=1}^{i*(d-1)}(1-(1-c)^{i*(\frac{(d-1)*s}{b}+l)+(k+j)*\frac{s}{b}+l})$ probability of a corresponding child failure, the balanced tree's completeness is:

$$N - \sum_{i=0}^{\log_d(N)-1} \frac{N}{(d-1)*d^i} * \sum_{j=1}^{d-1}(1-(1-c)^{j*\frac{s}{b}+l}) * (1 + \sum_{k=1}^{i*(d-1)} (1-(1-c)^{i*(\frac{(d-1)*s}{b}+l)+(k+j)*\frac{s}{b}+l})) \tag{6}$$

4.1.4 Supernode (Seaweed)

In a system with supernodes the nodes form a tree whose internal nodes replicate data before sending it up to the root of the tree. Typically the tree is balanced and has uniform degree d. To prevent the loss of data when an internal node fails, there are r replicas of each internal node. When a node receives data from a child it replicates the data on the replicas before replying to the child. Ideally an internal node can replicate data from one child concurrently with receiving data from another child. In order to prevent sending small amounts of data through the tree, a node typically batches data before sending them to its parent.

The analytical model allows internal nodes to replicate data while receiving new data. Also internal nodes send data to their parents as soon as they have received all data from their children. This means the model hides all but the initial delay in receiving the first bit of information ($\frac{s}{b} + l$) in the replication time ($\frac{r*d*s}{b} + 2*l$). Using this model, the completion time for supernode solution is:

$$\log_d(N) * (\frac{s}{b} + l + \frac{r*d*s}{b} + 2*l) \tag{7}$$

To simplify analysis the model assumes that there is enough replication to avoid losing all replicas of a supernode simultaneously. As a result, the only failures that affect completeness are leaf nodes who fail before sending data to their parents. The completeness is therefore:

$$\sum_{i=1}^{d} \frac{N}{d} * (1-c)^{i*(\frac{s}{b})+l} \tag{8}$$

4.1.5 Analysis

These models allow comparison of the completion time and completeness of the four techniques for aggregating data, as functions of the number of nodes and the size of the aggregated data. All other parameters are set to the values in Table 1.

In Figure 2 the completion time of each technique is shown as a function of the data size. The number of

	Description	Value	Source
N	Number of nodes	300,000	CorpNet[20]
b	Bandwidth	725Kbps	S^3 [29]
l	Latency	150ms	S^3 [29]
s	Data size	1MB	CERT[2]
c	Churn rate	$5.5 * 10^{-6}$	Farsite
r	Supernode replicas	4	Seaweed[20]
d	Node degree	16	Seaweed[20]

Table 1: Model parameters

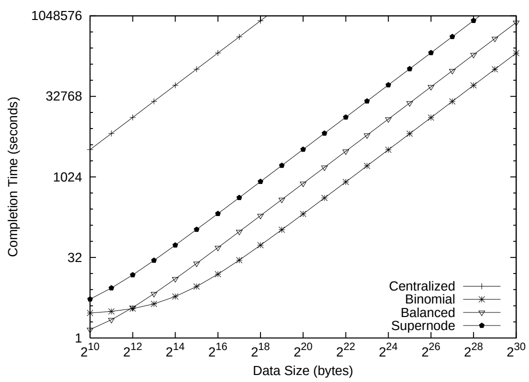


Figure 2: Completion Time vs. Data Size

nodes is fixed at 300,000. The completion times grow roughly linearly as the size of the data is increased. As expected, the centralized is the slowest, followed by the supernode technique (due to the replication cost). The balanced solution performs poorly once the aggregate data is over 64KB and it trails binomial trees by roughly a factor of four.

In Figure 3 the completion time is shown as a function of the number of nodes. The data size is fixed at 1MB. As expected the centralized technique grows linearly and takes by far the most time. The other techniques have logarithmic growth. The supernode technique is a little more than 4 times slower than the balanced tree for the same number of nodes (which is expected because the replication factor is four). The binomial tree is again about four times faster than the balanced tree.

Figure 4 shows the completeness of the techniques (expressed as the number of nodes not included in the result) as a function of the data size. All other parameters set to the values in Table 1. The completeness for the centralized technique drops off rapidly as the data size increases because the transfers are done sequentially and a longer aggregation increases the probability that the last nodes to transfer their data will fail before they do. Supernodes perform better than balanced trees because of the replication in the supernodes, but perform worse than binomial trees because leaf node failures still affect completeness. Binomial trees have superior completeness because for a failed node's data to be lost every

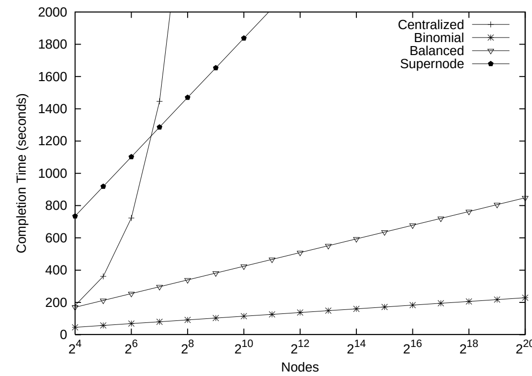


Figure 3: Completion Time vs. Nodes

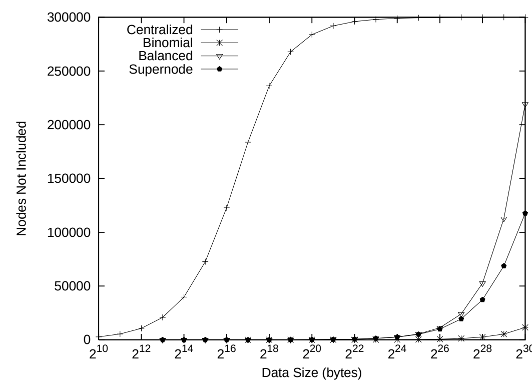


Figure 4: Completeness vs. Data Size

node whose tree includes the failed node's data also fail before exchanging their data with another node. As the aggregation proceeds the probability of this happening become vanishingly small.

Increasing the number of nodes also affects completeness. Since completeness is expressed as the number of nodes whose data are not included in the result, completeness should decrease at least linearly with the number of nodes because more nodes means more potential failures. As shown in Figure 5, completeness for the centralized technique drops off rapidly. The other techniques do quite well for under 1000 nodes and then decline. Binomial trees perform well even when there are over 1 million nodes in the network, with fewer than 50 nodes whose data are not included in the result.

4.2 Simulation

This section presents a discrete, event–based simulation of San Fermín that helps understand its scalability and reliability properties. The simulator was driven by measurements of real network topologies, and makes several simplifications in order to improve scalability and run time. In particular, the simulator builds the Pastry routing tables using global knowledge, does not include all of the

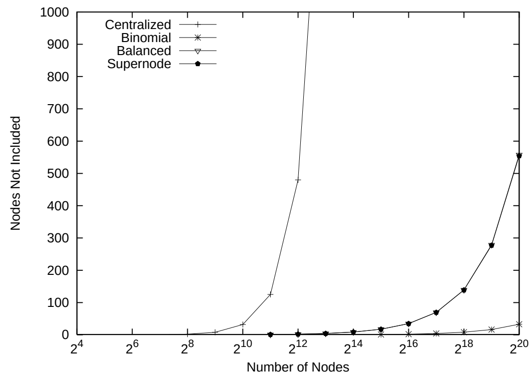


Figure 5: Completeness vs. Nodes

connection teardown states of TCP (as San Fermín does not wait for TCP to complete the connection closure), and does not model lossy network links. In Section 4.3 a prototype implementation of San Fermín is evaluated in a real world environment.

4.2.1 Experimental Setup

The simulations used network topologies from a Computer Science department and PlanetLab. The Computer Science department topology (CS) consists of a central switch connected to 142 systems with 1 Gbps links, 205 systems with 100 Mbps links, and 6 legacy systems with 10 Mbps links. Configurations with different numbers of nodes were constructed by randomly choosing nodes from the overall topology.

The PlanetLab topology was derived from data provided by the S^3 project [29]. The data provides pairwise latency and bandwidth measurements for all nodes on PlanetLab. Intra–site topologies were assumed to consist of a single switch connected to all nodes. The latency of an intra–site link was set to 1/2 of the minimum latency seen by the node on that link, and the bandwidth to the maximum bandwidth seen by the node. Inter–site latencies were set to the minimum latency between the two sites as reported by S^3 minus the intra–site latencies of the nodes. The inter–site bandwidths were set to the maximum bandwidths between the two sites.

4.2.2 Completion Time

The first set of experiments measures the completion time of San Fermín on the network topologies. The first experiment varied the number of nodes in the system to demonstrate the scalability of San Fermín; the results of the CS topology are shown in Figure 6. Each data point represents the average of 10 trials and standard deviations are shown. The completion time increases slightly as the number of nodes increases; when the number of nodes increases from 32 nodes to 1024 nodes the completion time only increases by about a factor of four.

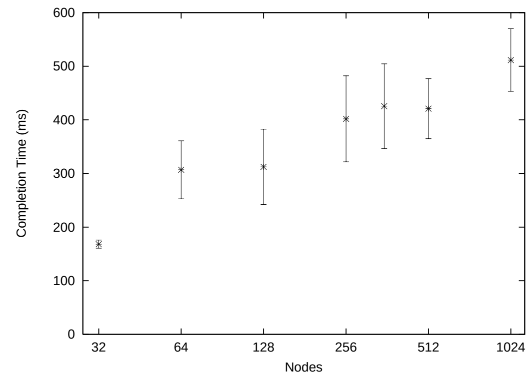


Figure 6: Completion Time vs. Nodes

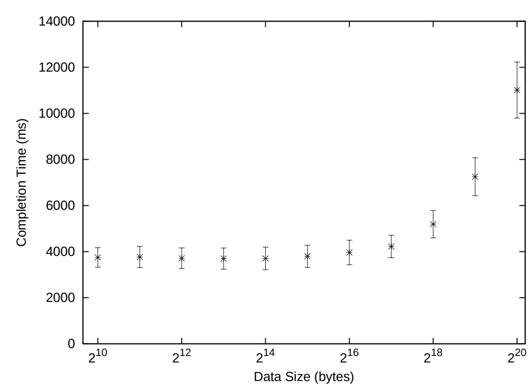


Figure 7: Completion Time vs. Data Size

Nodes	CS		PlanetLab	
	DHT	TCP	DHT	TCP
32	1.6	3411	2.8	2821
64	1.7	3428	3.4	2976
128	1.8	3565	3.6	3226
256	1.9	3741	4.1	3465
353	2.0	3831		
394			4.1	3488
512	2.2	3994	4.3	3604
1024	2.7	4011	5.5	3682

Table 2: Network Traffic (KB)

A 1024 node aggregation of 1MB completed in under 500ms. The PlanetLab topology (not shown) has similar behavior, as the completion time also increases by approximately a factor of four as the number of nodes increases from 32 to 1024.

Figure 7 shows the result of varying the data size while using all 394 nodes in the PlanetLab topology. Each data point represents the average of 10 trials and standard deviations are shown. The completion time is dominated by the DHT and message header overheads for data sizes under 128KB. When aggregating more than 128KB the completion time increases significantly. The CS topology (not shown) has a similar pattern in which all of the data sizes under 128KB take about 200ms and thereafter the mean time increases linearly with the data size.

4.2.3 Network Traffic

To evaluate the network traffic overhead of San Fermín, we ran experiments with different numbers of nodes, data sizes, and network topologies (Table 2). The traffic is segregated into that incurred by Pastry and that incurred by TCP. The first set of experiments vary the number of nodes. The increase in traffic is slight as the number of nodes increases. San Fermín on the PlanetLab topology has higher DHT and lower TCP traffic than on the CS topology. This is because PlanetLab's latency is higher and more variable, causing the overall aggregation process to take much longer (which naturally increases the number of DHT messages sent). The PlanetLab bandwidth is also highly variable (especially intra–site links versus inter–site links). This means that the variability in partnering time is very high, so that slow partnerings that would otherwise occur do not because faster nodes have already computed the answer.

The data size is the most significant contributor to traffic overhead. Figure 8 shows that doubling the size causes the traffic to slightly more than double. The standard deviation in traffic across different runs was less than 4%. The CS topology has a lower DHT traffic because fewer timeouts happen over the LAN. The DHT overhead does not significantly increase as the data size increases. The traffic per node increases roughly linearly

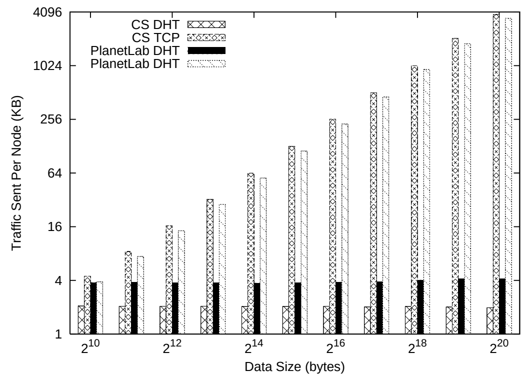


Figure 8: Traffic vs. Data Size

as the size of the data increases.

4.2.4 Completeness

The final set of simulations measured the effectiveness of San Fermín in the face of node failures. Failure traces were synthetically generated by randomly selecting nodes to fail during the aggregation. The times of the failures were chosen randomly from the start time of the aggregation to the original completion time. The responsiveness of the underlying DHT in noticing failures is varied to demonstrate the effect on San Fermín.

Due to the timeout mechanism in San Fermín, failures may be detected before the underlying DHT. As a result, the average completion time is less than the Pastry recovery time (Figure 9). On the PlanetLab topology, when the Pastry recovery time is less than 5 seconds, the cost of failures is negligible because other nodes use the time to aggregate the remaining information (leaving only failed subtrees to complete). When the recovery time is more than 5 seconds then some nodes end up timing out a failed subtree before continuing. The CS department topology (not depicted) typically completes in less than 500ms so all non–zero Pastry recovery increase the completion time. However, the average completion time is less than the Pastry recovery time for all recovery times greater than 1 second.

Figure 10 shows how failures affect completeness. Since failures occurred over the original aggregation time, altering the Pastry convergence time has little effect on the completeness (and so the average of all runs is shown). The number of failures has different effects on the PlanetLab and CS topologies. There is greater variability of link bandwidths in the PlanetLab topology, which causes exchanges to happen more slowly in some subtrees. Failures in those trees are more likely to decrease completeness than in the CS topology, which has more uniform link bandwidths and the data exchanges happen more quickly. In both topologies the completeness is better than the number of nodes that failed – in

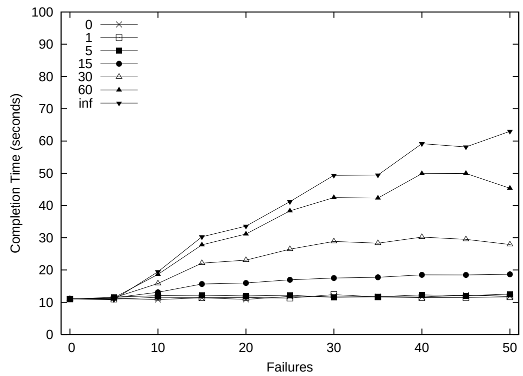


Figure 9: Completion Time vs. Failures

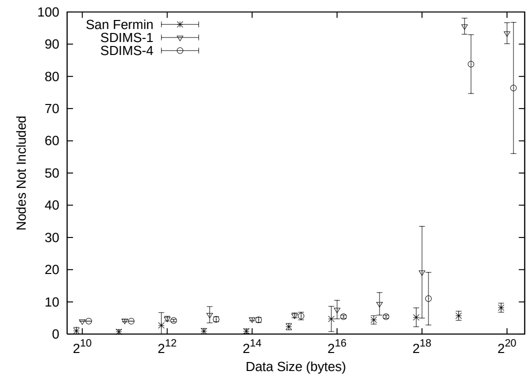


Figure 11: Completeness vs. Data Size

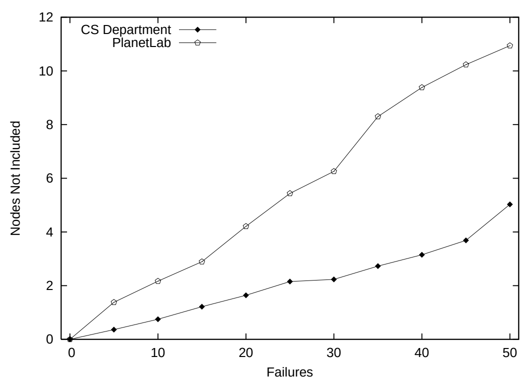


Figure 10: Completeness vs. Failures

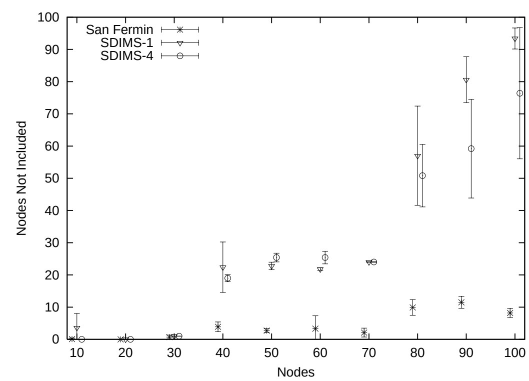


Figure 12: Completeness vs. Nodes

most cases a node fails after enough data exchanges have occurred to ensure its data are included in the result.

4.3 PlanetLab Results

This section presents the results comparing a San Fermín prototype and SDIMS [28] running on PlanetLab [21] . The San Fermín prototype is written in Java and runs on the Java FreePastry implementation. A set of 100 randomly selected live nodes with transitive connectivity and clock skew under 1 second was initially chosen. The same set of nodes was used for each test of that size. Subsets of nodes were chosen from the nodes which were alive at the end of the previous test. San Fermín and SDIMS tests were run consecutively. Nodes that could not be contacted between tests were automatically excluded from the smaller subsets. However nodes that failed to respond to some aggregation requests due to load were not explicitly excluded.

It must be noted that SDIMS was designed for streaming small amounts of data whereas San Fermín is designed for one–shot queries of large amounts of data. The comparison is performed primarily to demonstrate that existing techniques are inadequate for this task. One complication we encountered was zombie nodes in Pastry. San Fermín uses timeouts to quickly identify nodes that are unresponsive. SDIMS however, relies on the underlying DHT to identify unresponsive nodes, leaving it vulnerable to zombie nodes. After consulting with the authors, we learned that they avoid this issue on PlanetLab by building more than one tree (typically four) and using the result from the first tree to respond. In our experiments we measured SDIMS using both one tree (SDIMS–1) and four trees (SDIMS–4).

The experiments compare the time, overhead and completeness of SDIMS and San Fermín. A small amount of accounting information was included in each aggregation result to allow us to track which nodes are represented in the result. All tests were limited to 5 minutes in length. In SDIMS the results trickle up to the root over time. An SDIMS result was considered complete when either all nodes answered or at least half of the nodes answered and there was no update for 20 seconds.

4.3.1 Completeness

The first set of PlanetLab experiments evaluate the completeness of the algorithms as the amount of data increases (Figure 11). Each experiment used 100 Planet-

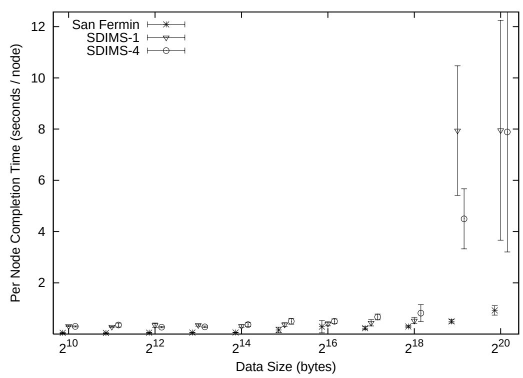


Figure 13: Per–node Completion Time vs. Data Size

Lab nodes and was repeated five times. The number of failed nodes is small for each algorithm until the data size exceeds 256KB. At that point SDIMS performs poorly because high–degree internal nodes are overwhelmed. San Fermín continues to include the answer from most nodes.

The next set of experiments measures how completeness is affected when the number of nodes is varied (Figure 12). The data size was fixed at 1MB. When there are few nodes SDIMS–4 and San Fermín algorithms do quite well. Once there are more than 30 nodes the SDIMS trees start to perform poorly due to high–degree internal nodes being overwhelmed with traffic.

4.3.2 Completion Time

Figure 13 shows per–node completion time, which is the completion time of the entire aggregation divided by the number of nodes whose data are included in the aggregation. This metric allows for meaningful completion time comparisons between San Fermín and SDIMS because they may produce results with different completeness. Data sizes larger than 256KB significantly increases the per–node completion time of SDIMS, while San Fermín increases only slightly. Although not shown, for a given data size the number of nodes has little effect on the per–node completion time.

Figure 14 shows the result of aggregating 1MB of data on 100 nodes for 50 runs of each system. San Fermín consistently provides high completeness and low completion time even in a dynamic environment like PlanetLab. SDIMS's performance is highly variable — SDIMS–1 occasionally has very high completeness and low completion time, but more often performs poorly with more than half of the runs missing more than 35 nodes from the answer. SDIMS–4 performs even worse with the all but 10 runs missing the answer from at least 80 nodes.

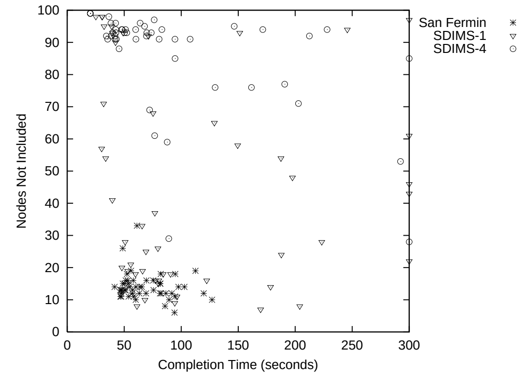


Figure 14: Completeness vs. Completion Time

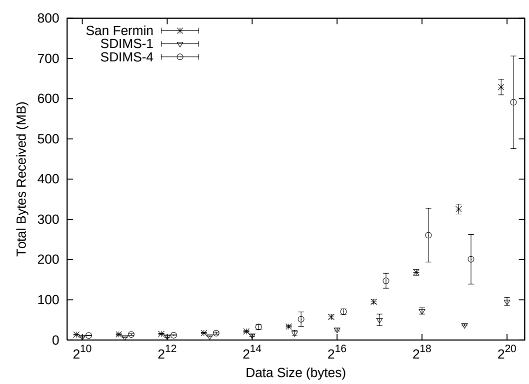


Figure 15: Traffic vs. Data Size

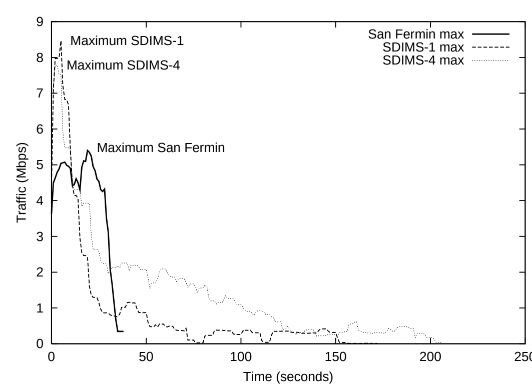


Figure 16: Peak Node Traffic

4.3.3 Network Traffic

Network traffic can limit the scalability of an aggregation technique. There are two considerations: the total traffic required by an aggregation; and the peak traffic seen by each node during aggregation. If the total traffic during an aggregation is too high the network infrastructure and other applications are likely to be impacted. If the peak traffic is too high then nodes in the system may become overloaded and fail. Figure 15 shows the impact different data sizes have on the total overhead in the system. For less than 256KB, SDIMS–1 incurs the smallest amount of overhead, followed by San Fermín and then SDIMS–4. After 256KB the overhead for SDIMS actually decreases because the completeness decreases. In this case nodes fail due to being overwhelmed by the traffic they receive. A single internal node failure causes the loss of all data for it and its children until either the internal node recovers or the underlying DHT converges.

Figure 16 shows the peak traffic received by any node during 1–second intervals. SDIMS internal nodes may receive data from all of their children simultaneously; the large initial peak of SDIMS traffic causes internal nodes that are not well–provisioned to either become zombies or fail. On the other hand, San Fermín nodes only receive data from one partner at a time, reducing peak traffic.

4.3.4 San Fermín Node Selection

An important aspect of San Fermín is that each node creates its own binomial aggregation tree. By racing to compute the answer high–capacity nodes naturally fill the internal nodes of the binomial trees, while low–capacity nodes fill the leaves and ultimately abort their own aggregations. The final experiment measures how effective San Fermín is at doing that. 1MB of data was aggregated from 100 PlanetLab nodes 10 times, and the state of each node when the aggregation completed was recorded. Table 3 shows the results, including the number of exchanges each node had to perform before completing its aggregation and the average peak bandwidth of nodes with the same number of exchanges remaining. Nodes with the higher capacity had fewer exchanges remaining, whereas the nodes with lower capacity aborted. The nodes in the middle tended to abort but some were still working. The average capacity of the nodes that aborted was 2.1Mbps, whereas the average capacity of the nodes still working was 3.2Mbps. This illustrates that San Fermín is effective at having high–capacity nodes perform the aggregation while low–capacity nodes abort.

Remaining	Aborted Nodes		Working Nodes	
Exchanges	Number	Mbps	Number	Mbps
0	0	0.0	38	4.3
1	0	0.0	105	3.9
2	0	0.0	116	3.6
3	9	2.5	56	2.3
4	82	2.0	32	2.2
5	143	2.0	19	1.2
6	107	2.4	9	1.1
7	62	2.0	1	0.8
8	14	1.7	0	0.0
9	16	2.4	0	0.0
10	3	1.6	0	0.0
11	0	0	0	0.0
12	2	1.9	0	0.0

Table 3: Node Progress

5 Related Work

Using trees to aggregate data from distributed nodes is not a new idea. The seminal work of Chang on Echo-Probe [7] formulated polling distant nodes and collecting data as a graph theory problem. More recently, Willow [27], SOMO [31], DASIS [1], Cone [3], SDIMS [28] Ganglia [19], and PRISM [13], have used trees to aggregate attributes. These systems build at most a single tree per attribute being aggregated. Willow, SOMO, and Ganglia build one tree for all attributes, whereas SDIMS, Cone, and PRISM build one tree per attribute. In contrast, San Fermín dynamically creates multiple trees for each attribute, improving aggregation performance and completeness especially when failures occur. San Fermín also differs from these systems in supporting one–shot queries with large results.

Seaweed [20] performs one-shot queries of small amounts of data and like San Fermín is focused on completeness. However, Seaweed trades completion time for completeness in that queries are expected to live for many hours or even days as nodes come online and return results. Seaweed uses a supernode based solution which further delays the timeliness of the initial results. Instead San Fermín focuses on a different part of the design space, robustly returning the results from living nodes in a timely manner.

CONCAST [4] implements many-to-one channels as a network service. In many respects it is IP multicast in reverse. It uses routers to aggregate data over a single tree. As the size of the aggregate data grows the memory and processing requirements on routers becomes prohibitive.

Gossip and epidemic protocols have also been used for aggregation [16, 10, 15, 14], perhaps the most well-known of which is Astrolabe [26]. Gossip and epidemic protocols are inherently imprecise – if the data are not replicated the result may be missing some, and if the data are replicated some data may be duplicated in the result, perhaps many times. In contrast, San Fermín ensures that data are represented at most once in the result, and uses dynamic binomial trees to improve completeness.

Data aggregation is also an issue in sensor networks.

Unlike our work, the major concerns in sensor networks are power consumption and network traffic. San Fermín instead focuses on completeness even in the face of network failures on more traditional network topologies. Examples of data aggregation in sensor networks are TAG [18], Hourglass [24], and Cougar [30].

Distributed query processing involves answering queries across a set of distributed nodes. The most relevant to our work are systems such as PIER [11], which stores tuples in a DHT as part of processing a query. Distributed query processing also encompasses performing queries on continuous streams of data. Aurora [8], Medusa [8], and HiFi [9] are examples. San Fermín is designed for one–shot aggregation, in future work we intend to examine reusing the resulting trees for the aggregation of continuous streams.

6 Conclusions

This paper presents San Fermín, a technique for aggregating large amounts of data that provides high completeness, low completion time, and is scalable. By having each node compute the aggregated result by creating its own binomial tree San Fermín naturally ensures that high–capacity nodes perform the bulk of the aggregation, while limiting the effect of node failures and slow nodes. Having each node compute the aggregation also makes San Fermín highly fault–tolerant since a node that does not fail will compute an aggregation of some subset of the nodes, perhaps only itself. San Fermín scales better than conventional techniques as the number of nodes or the data size increases, and reduces peak network traffic to prevent overwhelming nodes.

References

[1] K. Albrecht, R. Arnold, M. Gähwiler, and R. Wattenhofer. Aggregating information in peer-to-peer systems for improved join and leave. In *Peer-to-Peer Computing*, 2004.

[2] Anonymous. Details omitted for double-blind reviewing, Sept. 2006.

[3] R. Bhagwan, G. Varghese, and G. Voelker. Cone: Augmenting DHTs to support distributed resource discovery. Technical Report CS2003-0755, UCSD, 2003.

[4] K. Calvert, J. Griffioen, B. Mullins, A. Sehgal, and S. Wen. Concast: design and implementation of an active network service. *IEEE JSAC*, 19(3), 2001.

[5] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Exploiting network proximity in distributed hash tables. In *FuDiCo*, 2002.

[6] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8), 2002.

[7] E. J. H. Chang. Echo algorithms: Depth parallel operations on general graphs. *IEEE TSE*, 1982.

[8] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S. B. Zdonik. Scalable distributed stream processing. In *CIDR*, 2003.

[9] M. J. Franklin, S. R. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, E. Wu, O. Cooper, A. Edakkunni, and W. Hong. Design considerations for high fan-in systems: The HiFi approach. In *CIDR*, pages 290–304, 2005.

[10] I. Gupta, R. van Renesse, and K. P. Birman. Scalable fault-tolerant aggregation in large process groups. In *IEEE DSN*, 2001.

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

[12] F. l. F. J. Guicahrd and J. P. Vasseur. *Definitive MPLS Network Designs*. Cisco Press, 2005.

[13] N. Jain, D. Kit, D. Mahajan, M. Dahlin, and Y. Zhang. PRISM: Precision integrated scalable monitoring. Feb. 2007.

[14] M. Jelasity, W. Kowalczyk, and M. van Steen. An approach to massively distributed aggregate computing on peer-to-peer networks. In *Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 2004.

[15] M. Jelasity and A. Montresor. Epidemic-style proactive aggregation in large overlay networks. In *ICDCS*, 2004.

[16] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM TOCS*, 23(3):219–252, 2005.

[17] B. R. Liblit. *Cooperative Bug Isolation*. PhD thesis, University of California, Berkeley, Dec. 2004.

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

[19] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7), July 2004.

[20] D. Narayanan, A. Donnelly, R. Mortier, and A. Rowstron. Delay aware querying with Seaweed. In *VLDB*, 2006.

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

[22] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill Higher Education, 2000.

[23] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *ICDCS*, 2001.

[24] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh. Hourglass: An infrastructure for connecting sensor networks and applications. Technical Report TR-21-04, Harvard University, 2004.

[25] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *SIGCOMM*, 2001.

[26] R. van Renesse and K. Birman. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM TOCS*, May 2003.

[27] R. van Renesse and A. Bozdog. Willow: DHT, aggregation, and publish/subscribe in one protocol. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2004.

[28] P. Yalagandula and M. Dahlin. A scalable distributed information management system. In *SIGCOMM*, 2004.

[29] P. Yalagandula, P. Sharma, S. Banerjee, S. Basu, and S.-J. Lee. S3: a scalable sensing service for monitoring large networked systems. In *SIGCOMM workshop on Internet network management*, 2006.

[30] Y. Yao and J. Gehrke. The Cougar approach to in-network query processing in sensor networks. *SIGMOD*, 31(3):9–18, Sept. 2002.

[31] Z. Zhang, S.-M. Shi, and J. Zhu. SOMO: Self-organized metadata overlay for resource management in P2P DHT. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[32] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE JSAC*, 2003.