Design Considerations for High Fan-in Systems: The *HiFi* Approach

Michael J. Franklin*, Shawn R. Jeffery*, Sailesh Krishnamurthy*, Frederick Reiss*,
Shariq Rizvi*, Eugene Wu*, Owen Cooper*, Anil Edakkunni*, and Wei Hong+

*EECS Dept., UC Berkeley +Intel Research Berkeley

Abstract

Advances in data acquisition and sensor technologies are leading towards the development of "high fan-in" architectures: widely distributed systems whose edges consist of numerous receptors such as sensor networks, RFID readers, or probes, and whose interior nodes are traditional host computers organized using the principles of cascading streams and successive aggregation. Examples include RFID-enabled supply chain management, large-scale environmental monitoring, and various types of network and computing infrastructure monitoring. In this paper, we identify the key characteristics and data management challenges presented by high fan-in systems, and argue for a uniform, query-based approach towards addressing them. We then present our initial design concepts behind *HiFi*, the system we are building to embody these ideas, and describe a proof-of-concept prototype.

1. Introduction

Organizations across a large spectrum of endeavors are becoming increasingly dependent on the availability of accurate, targeted, and up-to-the-minute information about the status of their operations. This information provides real-time visibility into disparate phenomena, which can be used to monitor operations, detect problems, and support both short and long-term planning. Such visibility is enabled by continuing improvements in computing (e.g., wireless smart sensors) and communications (e.g., increasingly ubiquitous network connectivity).

1.1. Applications

In many cases, the phenomena being monitored exist in the *physical* world. For example, environmental monitoring using sensors is emerging as an area of great interest, where the phenomena being monitored include wildlife behavior, air and weather conditions, or seismic readings. Other physical monitoring applications are more closely tied to human organizations, such as the monitoring and control of supply chains, logistics, traffic, factories, pipelines, or energy usage.

In other cases, the phenomena being measured are *virtual*, such as network and computing infrastructure or application monitoring. Many emerging applications combine data from both worlds to provide increasingly detailed real-time models of complex, widely-distributed organizations and environments.

These applications span many different domains but they all share a dependence on a sophisticated computing and communications infrastructure to deliver data that will provide an accurate and actionable view of their domain. Such applications also vary widely in requirements, but in general, they all depend on the accuracy, timeliness, completeness, and relevance of data in order to support more effective decision making.

1.2. High Fan-in Systems – The "Bowtie"

We envision systems in which a large number (many thousands or more) of receptors exist at the edges of the network to collect raw data readings. For example, in a supply chain management deployment, collections of sensors and RFID readers on individual store shelves (in a retail scenario) or dock doors (in a warehouse/manufacturing scenario) continuously collect readings. These readings include "beeps" from low-function passive RFID tags indicating the presence of particular tagged objects (such as cases of goods), as well as more content-rich information from smart sensors and higher-function tags such as temperature readings and shipping histories.

---

*This work was funded in part by NSF under ITR grants IIS-0086057 and SI-0122599, by the IBM Faculty Partnership Award program, and by research funds from Intel, Microsoft, and the UC MICRO program.

Proceedings of the 2nd CIDR Conference,
Asilomar, California, 2005

These "edge" devices produce data that will be aggregated locally with data from other nearby devices. That data will be further aggregated within a larger area, and so on. This arrangement results in a distinctive bowtie topology we refer to as a *high fan-in system* (see Figure 1). A sophisticated system such as one supporting a nation-wide supply chain application may consist of many widely dispersed receptor devices and many levels of successively wider-scoped aggregation and storage. Such systems will comprise a disparate collection of heterogeneous resources, including inexpensive tags, wired and wireless sensing devices, low-power compute nodes and PDAs, and computers ranging from laptops to the largest mainframes and clusters.

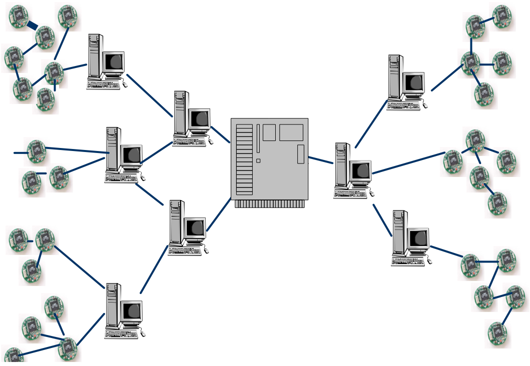

Figure 1 - The high fan-in bowtie

This hierarchical bowtie shape arises due to two main reasons. First, the sheer volume of raw data produced at the edges of a large system could easily overwhelm a flatter architecture, in terms of both bandwidth and processing costs. Data cleaning, filtering, and aggregation must be done as close to the edges as possible to minimize data handling requirements of the system as a whole. The bowtie shape lends itself naturally to an approach where computation is pushed out to the lowest common ancestor (LCA) of the edge nodes that are producing data used in any particular computation. Second, organizational concerns stemming from the geographic-oriented nature of many of our target applications and from the structure of the organizations that deploy these systems lend themselves naturally to a hierarchical structure.[1]

In many situations, of course, the topology will be much more complicated than that implied by Figure 1 above. For example, there will be cases where computations or data flows skip levels of the system, or there may be connections at various points in the network to external systems as would arise when multiple organizations choose to federate parts of their information infrastructure.[2] Nonetheless, our position is that the general notion of hierarchical structure and the ideas of successive aggregation and cascading streams (as presented in Section 3) that go with it are powerful concepts for organizing these complex systems, and where appropriate, can provide many advantages in programmability, ease of deployment, and efficiency.

1.3. Towards a Unified High Fan-in Framework

Today, the state-of-the-art in building high fan-in information systems is a piecemeal approach — a device-specific programming environment is used to task the edge receptors, a separate transport or information bus is used to route the acquired readings, and a database system or other data manager is used to collect and process the them. As a result, high fan-in deployments have tended to be costly, difficult, and inflexible.

In contrast, our work is based on the notion that stream query processing and streaming views can serve as a *unified declarative framework* for data access across an entire high fan-in environment. As we discuss in Section 3, stream-oriented queries can be used to accomplish many of the data manipulation tasks required in a high fan-in system, including data cleaning, event monitoring, data stream correlation, outlier detection, and of course, aggregation.

Our work builds on the growing body of work in the areas of data stream processing, sensor network databases, and data integration, but it also addresses a number of challenges that arise from the unique properties of high fan-in architectures and the applications they support. From a data management perspective, the most challenging new aspect that high fan-in systems bring to the table is the wide range they span in terms of three key characteristics: time, space, and resources.

Time – Timescales of interest in a high fan-in system can range from seconds or less at the edges, to years in the interior of the system. At the edges of a high fan-in system are receptor devices that repeatedly measure some aspect of the world. These devices are typically concerned with fairly short time scales, perhaps on the order of seconds or less.

As one moves away from the edges, the timescales of interest increase. For example, in a retail RFID scenario, individual readers on shelves may read several times a second, while the manager of a store may be concerned with how sales of particular items are going over the course of a morning, and planners at regional and corporate centers may be more concerned with longer-term sales trends over a season or several seasons.

---

[1] See the supply chain scenario described in Section 2 for an example of this.

[2] Such federation, for example, is envisioned in the standards for supply chain information sharing being proposed by the EPCGlobal organization [19].

Space - As with time, the area of geographic interest grows significantly as one moves from the edges of a high fan-in system to the interior. Again using the retail RFID scenario, individual readers are concerned with a space of a few square meters, aggregation points within the store would be concerned with entire departments or perhaps the store as a whole, and regional and national centers are concerned with those much larger geographical areas.

Resources – Finally, the range of computing resources available at various levels of a high fan-in system also vary considerably, from small, cheap sensor motes on the edges, up to mainframes or clusters in the interior of the system. Communication resources also can range from low-power, lossy radios at the edges, to dedicated high-speed fiber in the interior.

In addition to the fundamental issues that arise from the issues of scale along these three dimensions, there are many other technical challenges to be addressed. These include fundamental questions about how best to optimize and run queries in the network; how to process queries using views over streaming data that involve aggregation, hierarchies, and time windows; how to archive detail data at various points in the system; and how to build an infrastructure that is easy to deploy, manage, and adapt.

1.4. Paper Overview

In the remainder of this paper, we present a current snapshot of our work on high fan-in systems. Specifically, we describe the design considerations for the *HiFi* system, which is currently under development at UC Berkeley. We present the overall philosophy behind HiFi, detail our initial system's architecture and query processing approach, discuss some of the open issues we are beginning to address, and give an overview of our initial proof-of-concept prototype built using the TelegraphCQ adaptive data stream processor [12] and the TinyDB sensor network database system [28]. First, however, we describe an example high fan-in environment in more detail.

2. A Motivating Scenario: Supply Chain Management

As stated in the introduction, there are numerous application scenarios that exhibit a high fan-in topology. In this section, we briefly describe one such application, Supply Chain Management (SCM). SCM presents a particularly compelling case for high fan-in systems for several reasons. First, emerging SCM systems are large-scale systems that can span national or even global distribution networks. These systems have natural aggregation points resulting in the characteristic hierarchical, high fan-in structure. Second, there is now widespread recognition in industry of the cost savings and efficiency gains to be had by exploiting accurate, up-to-the-minute information throughout the supply chain. Third, the impending availability of RFID and related technology along with mandates from large organizations including Wal-Mart, the DoD, and the FDA are driving tremendous interest in real-time SCM solutions.

The promise of RFID-enabled supply chains is that organizations will have complete, accurate, and timely visibility into all aspects of their supply chain, from suppliers, manufacturing, distribution, sales, and even return of goods. Such visibility can enable a wide range of applications, including "track and trace" of individual sales units (e.g., cases or cans of soft drinks, vials of pills, etc.), accurate replenishment scheduling, and more efficient shipping and receiving. Accurate information can also greatly enhance planning and monitoring at all levels of an organization and across organizations.

At the base of an RFID-enabled supply chain are passive EPC (Electronic Product Code) tags attached to pallets, cases, or individual units [19]. These tags are able to transmit a 96-bit EPC code that identifies the manufacturer, type of item, and a unique serial number. More sophisticated tags can provide additional information such as sensor readings or transportation history. These tags are remotely sensed by RFID *readers*. Readers are placed at numerous locations in the supply chain, such as manufacturing lines, loading dock doors, forklifts and trucks, store shelves and store checkouts.

In a typical supply chain, a single tag is likely to be scanned 10 to 15 times. These readings (called "beeps") represent, in aggregate, a potentially huge data stream. Furthermore, current RFID technology is inherently error-prone, meaning that these streams may contain a significant amount of dirty data. The solution to both of these problems is to process the raw beeps to remove erroneous data and to reduce the volume through aggregation and event detection.

For the purposes of this paper, we focus on the "back-end" of an SCM system, from suppliers to the back rooms of stores. As an example, consider the following levels (see the left half of Figure 2):

- **RFID Readers** – the edges of the system consist of the readers that interrogate the tags and collect their values. Future SCM deployments also envision the use of sensors and sensor networks to provide additional monitoring capabilities.

- **Dock Doors** – due to limitations on the range and orientation at which tags can be read, multiple readers (or antennae) are arranged around a single door to reduce the probability of missed tags.

- **Warehouse** – the information from all the readers in the warehouse is aggregated in a computer (possibly a cluster) located in the warehouse.

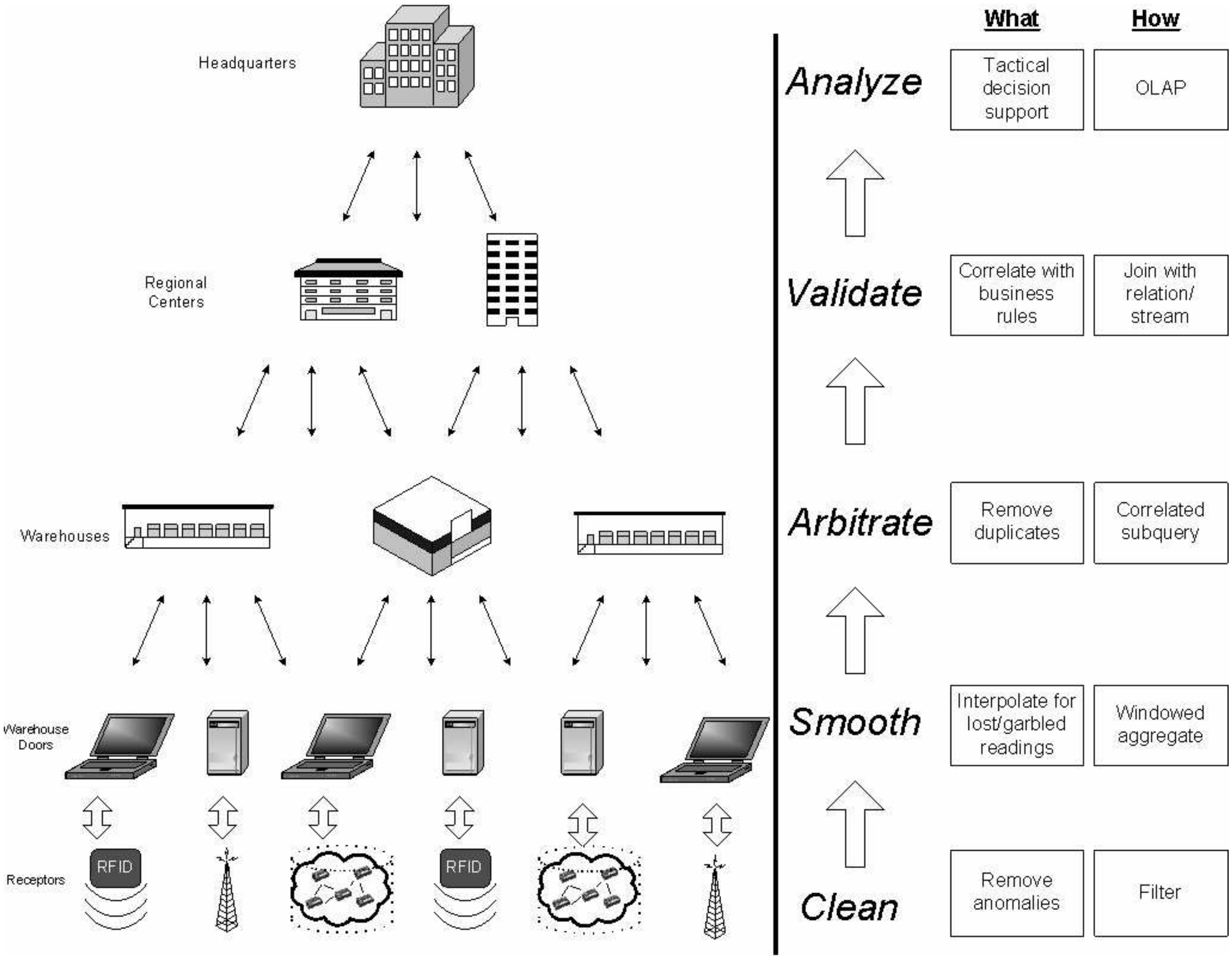

Figure 2 – High fan-in system levels with associated CSAVA processing stages in an SCM Environment

- **Regional Center** – a location where logistics can be collected and managed for an entire geographical region, for example, the Southwestern United States.
- **Headquarters** – the main location or data center for the corporation. For example, Wal-Mart does its corporate data processing in Bentonville, AR.

Such a supply chain is a high fan-in system, where data collected at the edges is successively cleaned, refined, and aggregated. As can be seen, this scenario exhibits the wide range of scales in terms of time, geography, and resources typical of high fan-in systems. We use this example throughout the rest of the paper.

3. Cascading Streams in HiFi

The HiFi system provides data management infrastructure for high fan-in environments. HiFi implements a *uniform declarative framework* for specifying data requests and driving system functionality. A stream-oriented query language is used to specify data requests and to describe logical views of the data available in various parts of the system. The use of successive stream-oriented query processing at each level of the system results in a flow of data from the edges inwards. We refer to this data flow as *cascading streams*. Note that this uniform approach stands in contrast to the current state-of-the-art, where each level of the system presents its own distinct API. HiFi uses stream-oriented query languages across all levels, which simplifies programming and enables a wide range of optimizations.

3.1. Uses of Queries

Although stream-based queries may not be suitable for all data processing tasks in HiFi, we believe that they can accomplish a wide array of tasks in a high fan-in environment. We list these tasks roughly in order from the edges to the interior of the system. At the edges, transformations are chiefly involved with making sense of what the receptors produce; they typically become more complex as data moves towards the interior.

Data Cleaning – Sensors and RFID readers are notoriously noisy devices, and dealing with the poor quality of data they produce is one of the main challenges in a high fan-in (or any receptor-based) system. We use declarative queries to specify cleaning functionality for single devices as well as across groups of devices.

Detecting Faulty Receptors – In addition to being noisy, receptors sometimes just fail. Depending on the actual failure mode (fail stop or fail dirty) and past history, it might be possible to detect a faulty sensor.

Conversion and Calibration – In order to produce meaningful measurements, physical sensors typically require calibration. Such calibration can be quite sophisticated and may need to be done on a continuous basis. Furthermore, the raw data produced by physical sensors must often be converted into units that are meaningful to a given application. Queries may be used to do simple conversion and calibration.

Outlier Detection – In many monitoring systems, expected events are of less immediate interest than anomalies. Queries can be used to detect and propagate various types of outliers in a streaming environment.

Data Aggregation – While the raw data from a single receptor may not be a high volume stream, processing raw data from thousands of receptors is unsustainable. Queries that have a wider *scope of interest* must necessarily summarize raw data in the form of coarse-grained aggregate histories.

Stream Correlation – Queries are also useful for comparing and correlating data from multiple streams. Such streams may be homogeneous, as in the case of temperature readings from a group of identical sensors, or heterogeneous, as in the case of combining temperature readings with RFID "beeps".

Complex Event Monitoring – One of the main functions of HiFi is to continuously monitor the environment for interesting events. Such events are not limited to simple events at the edge. Rather, they also include composite events described in terms of widely varying timescales and geographic areas. A streaming query language, suitably extended with event processing constructs, can be used to describe these events.

3.2. Multi-level Query Processing

In this section, we illustrate the power of successive processing of cascading streams through an example that we term "CSAVA" (pronounced "Cassava"). CSAVA consists of five core stages of processing (*clean*, *smooth*, *arbitrate*, *validate*, and *analyze*) designed to translate raw receptor readings into useful data for driving business processes. Figure 2 depicts CSAVA for our SCM example. At the bottom of the figure, RFID readers feed into processing nodes on warehouse doors which, in turn, feed into the main node at the warehouse. These local nodes aggregate and send their data to regional centers and so on. At each level, the nodes export views that perform successively more sophisticated functionality.

Receptor level: Cleaning

Each receptor itself performs the first step in data processing and *cleans* the stream by filtering anomalous readings that do not have a signal strength higher than some threshold `strength_T`:

```
CREATE VIEW cleaned_rfid_stream AS
(SELECT receptor_id, tag_id
  FROM rfid_stream rs
  WHERE read_strength >= strength_T)
```

Dock door level: Smoothing

Smoothing is the process of interpolating to compensate for lost readings and discarding anomalous readings by running a windowed aggregate (in this case, a `count`) over the cleaned stream. In this example, readings that have been seen at least count_T times in a window[3] are considered legitimate; others are dropped:

```
CREATE VIEW smoothed_rfid_stream AS
(SELECT receptor_id, tag_id
 FROM cleaned_rfid_stream
      [range by  ‘t1’, slide by ‘t2’]
 GROUP BY receptor_id, tag_id
 HAVING count(*) >= count_T)
```

Warehouse level: Arbitration

The tags reported after smoothing are those that a reader is reasonably sure to have seen. Multiple nearby readers, however, may have seen the same tag. To avoid over-counting or other inaccuracies, data from multiple readers must be *arbitrated* to determine where the product corresponding to the tag actually is located. In this example, the system also aggregates what each node has seen before passing the stream to the higher levels:

---

[3] Note that we use "range by" to specify the width of the window and "slide by" to specify its movement.

```
CREATE VIEW product_counts AS
(SELECT receptor_id,
        count (distinct tag_id)
 FROM smoothed_rfid_stream rs
      [range by ‘t3’, slide by ‘t4’]
  GROUP BY receptor_id, tag_id
  HAVING count(*) >= ALL
        (SELECT count(*)
         FROM smoothed_rfid_stream
         [range by ‘t3’, slide by ‘t4’]
         WHERE tag_id = rs.tag_id
         GROUP BY receptor_id))
```

Regional center level: Validation

At this point in the hierarchy, the `product_counts` stream contains an aggregated view of products seen at each warehouse or store. With this data, the regional center can use known business rules such as “I know that the warehouse in Springfield should have 10,000 widgets” to *validate* that the supply chain is behaving as expected.

Headquarters level: Analysis

Once the high-level business behavior is determined, headquarters can *analyze* this data through data mining-type query operations to understand how the supply chain is behaving. Note that this is done in real time to drive organizational decision-making.

3.3. CSAVA Discussion

CSAVA processing can be generalized to handle data from other types of receptors in other applications. In general, *clean* involves operations over a single data item, *smooth* occurs over a window of data items from a single receptor, and *arbitration* occurs over streams from multiple receptors. For example, *cleaning* for a sensor network application involves filtering individual readings that do not make sense (i.e., a negative sound reading) or are not interesting to the application, while *arbitration* entails comparing values from multiple sensors in the same area for calibration and outlier detection.

In addition to the steps outlined above, there are a variety of auxiliary tasks that may occur throughout the CSAVA process[4]. Data values retrieved from RFID readers may need to be *converted* from their raw RFID code to some organizationally meaningful handle such as product ID. Additionally, there may be organizational information, such as tracking history, which can *augment* the bare product ID. Finally, *aggregation*, both in time and space, occurs throughout this process, whenever the raw data is not needed or bandwidth is scarce.

In this example, each task in CSAVA was placed at a reasonable location in the hierarchy; in practice, however, the placement of each stage in CSAVA is flexible, provided that the appropriate data is available. For example, *validation* may occur lower in the hierarchy if the validating data is also pushed down (e.g., the system could push a static relation containing expected RFID tags to the edge of the hierarchy). We address the issue of operator placement in more depth in Section 4.3.

4. HiFi Design Concepts

In the previous sections we laid out the motivation, applications, and the cascading stream processing model for high fan-in environments. Given this background, we now describe the key aspects of our emerging design for the HiFi system.

4.1. Hierarchical Windowed Views

One consequence of our decision to use stream-oriented queries as the common API throughout HiFi is that the nodes at various levels of the system can expose the data they provide using *views*. Using views to structure distributed systems has long been studied in the data integration and federated database literature [18][25][32], and many of the techniques developed there can be used in HiFi. There are, however, several aspects of high fan-in architectures that push the envelope of this technology.

First, the views in HiFi are typically over *streaming* data and the queries are *continuous*. In such systems, *windows* play a crucial role in both query processing and semantics. Window specifications divide unbounded streams into finite collections of data items over which queries can be executed. Windows are specified by *range* and *slide* parameters (typically expressed in time or tuples). The first parameter specifies the width of the window; the second specifies how the window moves as time progresses or as new tuples arrive.

The definition and use of streaming views with windows is still an open problem. When the slide/range of a query does not precisely match that of a view it is not immediately obvious how or even if the view can be exploited. For instance, exploitation is generally possible if the range of the query matches (or in some cases, is subsumed by) that of the view and the slide of the query is a multiple of that of the view, or possibly if the slides are relatively prime. This has the flavor of periodic data processing [6].

Second, because *aggregation* is such a fundamental concept in HiFi, the views at each level of the system will often contain aggregates. When a query’s range exceeds its slide, the windows are *overlapping*. Evaluating aggregate queries over overlapping windows is challenging because input tuples must participate in multiple separate aggregate computations. We are

---

[4] Note that these additional steps all begin with a “C” or “A”, thus preserving our CSAVA acronym.

working on techniques to efficiently share the execution of periodic overlapping windowed aggregates.

Third, the *hierarchical* nature of the applications to be supported by HiFi emphasizes the issues of *granularity* and *scope*. As stated above, we expect that in general, the granularity of requests will become coarser and the scope larger as one moves from the edges towards the interior of a high fan-in topology. Aggregation and union operators can be used to achieve this; however, cases that do not follow this anticipated pattern are more difficult to handle. For example, privacy and security constraints may restrict the detail and scope of information allowed to be passed to some other node in the system.

4.2. Topological Fluidity

Another important issue in the design of HiFi is the rigidity of the connections between nodes in the system. In some levels of a high fan-in system, a hardwired topology may be natural. For instance, it makes sense for a node keeping track of items on a shelf to be hardwired to talk to the store's node, which in turn, is likely to talk to the regional node, and so forth. In such an arrangement, each node has a static parent (or small set of parents) and a relatively small static set of children with which it communicates. With static connections at all levels, nodes require only a small amount of state to keep track of the other nodes with which they communicate. Furthermore, both query and data flow are greatly simplified in a static system, as there are only a small number of paths through the system.

It is desirable in many cases to have more fluid connections between interior nodes. In such a topology, nodes would still be grouped into levels, but connections to parents (for data flow) and children (for query flow) would occur on an ad-hoc basis. Thus, the system can respond to runtime conditions by adding, removing, or changing links. Through fluid interior links, the system can route around overloaded or failed nodes and links, thus providing load balancing and fault tolerance.

Furthermore, some components in a high fan-in system, such as mobile nodes, do not fit into a static topology or may be disconnected for periods. For instance, a node mounted on a supply truck driving between distribution centers must have the ability to dynamically switch parents *en route* as well as be able to support disconnected operation. Finally, fluidity enables more fine-grained privacy and security provisioning. An organization can specify exactly which queries and data flows can go where on a flow-by-flow basis.

Of course, dynamism presents many challenges, including metadata management and query planning. For HiFi, we are developing a hybrid approach, where there are preferred wirings between nodes, but where alternate routes through the system can be used in response to runtime conditions.

4.3. Query Planning and Data Placement

Once a query is submitted to HiFi, it must be planned and disseminated before it can be run. Query planning in a high fan-in system involves a wide range of tasks. First of all, the system must identify the relevant data streams and determine the responsible receptors. If data is not already flowing from these receptors, the system must initiate data collection with appropriate settings (sample period, for instance). The system must determine the general flow of the data from the leaves and decide upon the operators needed to process, split, merge, and transform streams. Finally, the system must employ participating nodes to run these operators as data flow up the tree.

A key efficiency consideration in HiFi is the placement of queries and data across the nodes of the system. Given a query with a set of operators, the query planner must determine where in the hierarchy to place each operator. This decision attempts to reduce overall system bandwidth usage by pushing operators down the hierarchy. Some data streams (or static relations) may not be visible at lower levels of the hierarchy. Thus, the query planner should tend to push operators to the lowest level at which the streams and relations it operates over are visible.

Furthermore, the query planner must consider existing queries and data flows in order to exploit shared processing. For instance, if multiple operators from different queries process the same underlying data stream, then it may be advantageous to pull the operators up. Alternatively, it may be possible to improve the visibility of some queries by pushing streams or static relations down the hierarchy. This incurs initial bandwidth costs and complexity due to replication, but may improve parallelism and utilization of resources, and could provide overall bandwidth savings. Caching can also improve performance, but query and data placement in a cache-based system are inherently inter-dependent [24].

In a high fan-in system, the manner in which this query planning takes place may be done in a variety of ways, ranging from completely centralized to fully distributed. The simplest approach is to fully centralize the planning decisions. A new query would be sent to a single query planning node that has global knowledge and is able to fully plan and then disseminate the query. This approach is suitable for fully trusting organizations with relatively static data, query, and network characteristics.

Alternatively, a recursive approach, where the planning and dissemination phases are combined, may be more applicable. In this case, a query is introduced at some location, which becomes the root of the hierarchy for that query. The query then propagates from this point to the data sources one level at a time. At each step of the process, the current node plans its portion of the query using only knowledge of its immediate children. As we discuss in Section 5.4, we are implementing a flexible planning and optimization approach that follows this recursive query planning paradigm.

4.4. Event Processing

An important use case for HiFi involves the real-time monitoring and management of large distributed organizations such as supply chains. For such applications, it is necessary that the system enable the delivery of important status information and events in a timely fashion.

An *event* is defined as any significant occurrence in the system. A user may be interested in a variety of simple events over streaming data:

- A *taken-out-of-store* event may be defined simply as seeing a tuple on a particular data stream originating from an RFID receptor located at the exit of a store.
- A *fire-in-room* event may be defined as a simple "filter event" which is detected when a tuple with a temperature value more than 100°C is seen on a data stream.

C*omplex events* may also be of interest to a user. Unlike simple events, these require the joining and aggregation of multiple streams under intricate notions of time, ordering, and negation. Examples include:

- A *shoplifting* event may be triggered when the *taken-out-of-store* event is seen for an item WITHOUT the occurrence of the *purchased-at-counter* event for that item.
- A *person-in-danger* event may be triggered when the simple *fire-in-room* and *person-in-room* events are seen for the same room within a 10 second window.

In addition to real-time scenarios, event specifications may span large scales of time and space (e.g., a CEO who wants an alert when the nation-wide sales of a product in the previous week goes below a threshold).

In a system like HiFi, event and (SQL-based) data processing need to be done in a unified manner. However, SQL does not provide natural ways of expressing queries over ordered data (like time-ordered data streams) [26][36]. Hence, specification of complex events over streams needs to be done in a language that provides user-friendly ways of expressing ordering and negation, in addition to other constructs. Unified support for data and event processing is achieved in HiFi by extending the query language available to the user in a manner suitable to express complex event queries on data streams.

HiFi handles event processing using state machine-based operators in the core data stream processing engine along with traditional relational operators. These new operators maintain and update state for event queries as they see tuples on different data streams (possibly coming from other operators). They trigger an event when a transition to an accepting state is made. The output of these operators can be further processed by other data operators, giving a unified framework for event and data processing.

4.5. Archiving and Prioritization

In many situations, in addition to real-time information, there is also a need (or at least a desire) to have access to the underlying detail information, perhaps in a delayed or archival fashion. Examples include data mining and long-term planning applications as well as regulation-driven requirements, such as those arising from Sarbanes-Oxley compliance [39].

We are designing HiFi to support a spectrum of delivery requirements, spanning the range from real-time delivery of status and event notifications to background delivery and archiving of detail information. The basic approach is: "send summaries, anomalies, and alerts first; the details can follow later".

HiFi meets these varied delivery requirements through a dynamic prioritization architecture. The first priority of HiFi's scheduling subsystem is ensuring that archival data is not permanently lost [13]. Each node must ensure that its archival data is eventually delivered to a node with permanent storage. Nodes without local storage keep a buffer of recent data and send the contents of this buffer to remote nodes as needed.

Once HiFi has ensured the integrity of archival data, it devotes time to the other types of data. For these data, HiFi employs Data Triage [37] to provide the highest quality of timely results possible given the resources remaining. If there is time to process all data relevant to a monitoring query, HiFi will do so; otherwise, the system will shed load by summarizing data that it does not have time to process and sending these synopses in place of the original data. The system reconstructs complete query results by combining computations on complete data with computations on synopses.

4.6. Real World Data

Perhaps the most unique challenge presented by high fan-in environments is the need to seamlessly integrate the physical world with the digital world. However, the characteristics of each realm differ greatly. Real world data can be seen as an infinite collection of unbounded continuous streams with loose semantics, whereas the digital world is inherently discrete (for our purposes, it is tuple-oriented) with strict semantics and guarantees. Furthermore, data collection techniques are imperfect at best and provide only a flawed glimpse of the real world. Physical receptor devices can introduce significant complexity due to their wide variance in terms of interface, behavior, and reliability. Thus, a challenge facing any receptor-based system is to bridge these disparate worlds in a manner that enables users of the system to both trust and make sense of the data the system provides.

Towards this end, HiFi uses *virtual devices* to interact with the physical world. A virtual device interfaces with multiple raw receptors that are in close proximity, processing and fusing their streams to produce more useful, higher-quality data. It does this by incorporating CSAVA-like processing, conversion and calibration, virtualization, lineage tracking, and quality assessment. Thus, a virtual device may combine declarative query processing with non-declarative processing, such as with soft sensors [34].

One of the more important services a virtual device provides is the support for the notions of answer quality and lineage. To provide for the first component, the virtual device augments receptor-based data with error estimates and confidence intervals. For example, a virtual device for a sensor network can use known techniques [20] for determining answer quality based on probability distributions. Similar methods apply to other receptors, provided that the error characteristics are known. Lineage is also tracked for each data item (or set of items) as it is processed within the virtual device.

A virtual device provides a rich interface to HiFi for interacting with the receptor (or set of receptors). It exposes an interface that consists of a suite of virtual streams, including multiple levels of processed data streams (ranging from raw to fully cleaned data), quality streams, and lineage streams. HiFi interacts with a virtual device by querying and correlating these streams to produce useful information. For instance, to determine the quality of a certain data value, HiFi would correlate a data stream with the corresponding quality stream. Thus, the virtual device exports an interface that is richer and more useful than a cleaning view over the raw data.

A virtual device provides other services as well, such as archiving, prioritization, actuation, and receptor management. All of these services are exposed via this same stream-based interface. For instance, a query over the archive "stream" allows access to past data.

We are currently exploring the extent of a virtual device's functionality and defining its behavior in various environments.

4.7. Privacy and Access Control

Privacy of data is a prime concern in environments where the flow of information crosses organizational boundaries. This is another case where the use of views to express exported data plays a role. Each HiFi node exports a particular set of views to the higher level nodes based on the access control policies specified by the node's organization. The query planner ensures that only those queries that can be written on top of these views are executed on that node. This restriction ensures that no information available at the node is leaked to the higher levels in an unauthorized manner.

The use of SQL views for specifying authorization policies and enforcing access control by query rewriting using views has been discussed for the centralized case in [38]. For HiFi, we can extend the approach to a distributed scenario in which authorization views are exported by distributed data sources.

4.8. System Management

Finally, a major requirement and challenge for the deployment of a large, integrated, distributed system such as HiFi is the ability to continuously monitor the state of the system itself and adaptively adjust its behavior. Furthermore, the system must be easy to modify in terms of the addition and removal of new components and types of components. While we are only beginning our investigation into the system management issue, we intend to exploit the fact that HiFi is itself a hierarchical system for monitoring and managing phenomena in hierarchical environments. Thus, we expect to use the HiFi infrastructure itself to accomplish much of the system management task.

5. Initial Architecture and Service Design

Having outlined the major design issues for HiFi, we now present a description of the initial system. We detail the functionality and services provided by HiFi by outlining its major components: the Metadata Repository (MDR), the Data Stream Processor (DSP), and the HiFi Glue.

5.1. Metadata Repository

The Metadata Repository (MDR) serves as a globally accessible catalog for system-wide information. This metadata is of three types: schema, views, and system information.

The schema contained in the MDR is the mediated schema of the system over which all application queries and views are written. It is assumed that this changes very infrequently. For instance in our SCM example, the mediated schema consists of sensor and RFID data.

The views stored in the MDR are those exported by each node in the system. The MDR also maintains a mapping of the views exported by a node and its physical location, which is vital for supporting a fluid, loosely-coupled topology.

The system information contained in the MDR includes node capabilities, authorization and privacy controls, and information relating to organizational boundaries and administrative domains. Additionally, the MDR maintains runtime information, such as the current set of queries running on each node, current network usage, and unavailable/unreachable nodes to help guide and optimize system behavior.

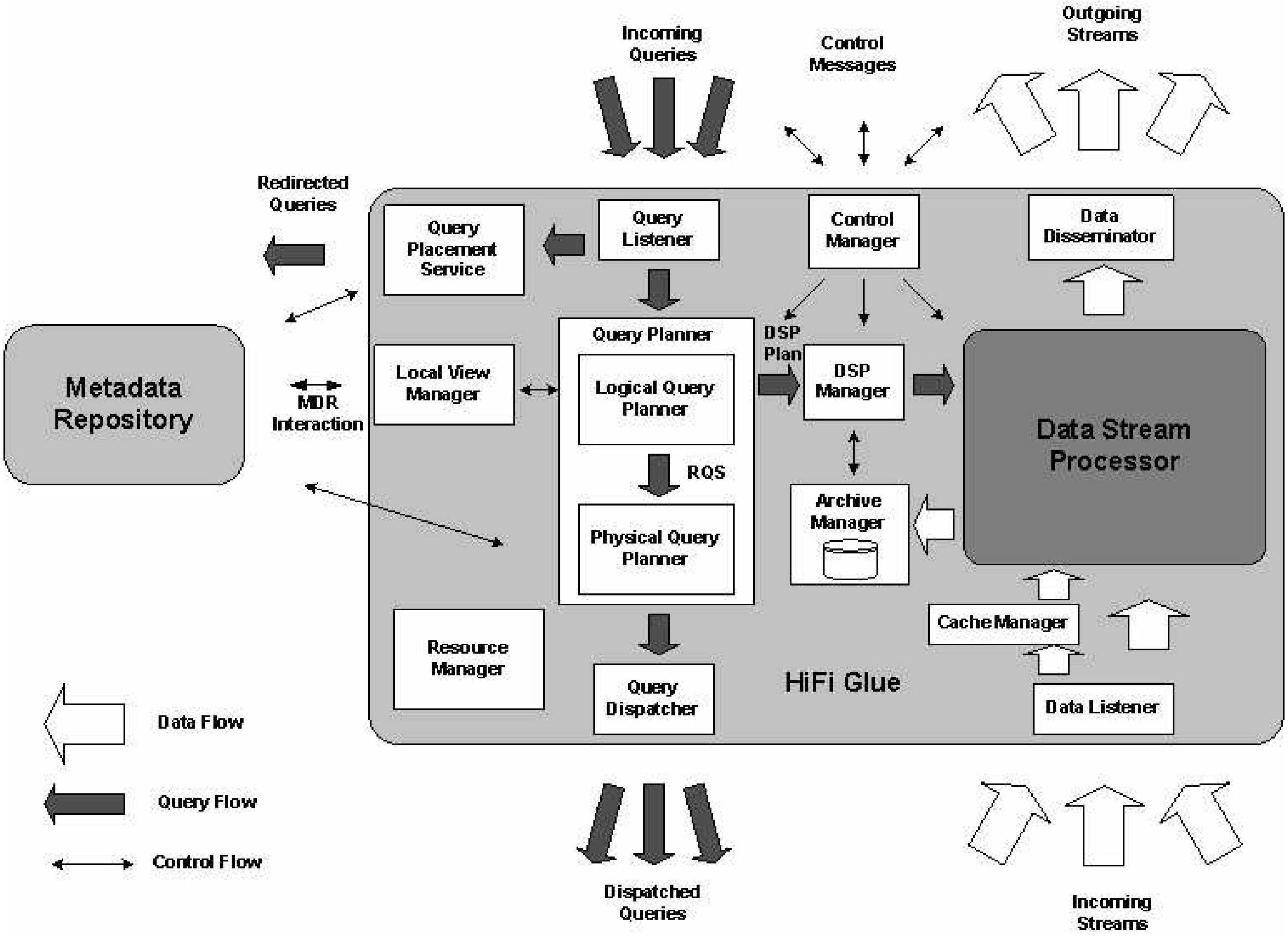

Figure 3 - Internal architecture of a HiFi node

The MDR can be implemented in a variety of ways, from fully centralized to fully decentralized, and this is a topic we are currently investigating.

5.2. Data Stream Processor

The Data Stream Processor (DSP) lives entirely within a HiFi node and is responsible for all single site data stream processing. Only the following simple functionality is expected of a DSP:

1. The ability to process continuous queries
2. The ability to add continuous queries on-the-fly
3. The ability to add sources on-the-fly
4. The ability to cancel queries

Additionally, when present, a HiFi node can profitably exploit:

1. The ability to modify a currently running query
2. The ability to suspend a currently running query
3. The archiving of streams
4. The querying of archived data

The DSP is oblivious of HiFi and could (in principle) be any stream processor such as TelegraphCQ [12], Aurora [1] or STREAM [30][5].

5.3. HiFi Glue

The HiFi Glue, which runs on each HiFi node, is the fabric that seamlessly binds together the system. It coordinates its local DSP, communicates with other HiFi nodes, and manages incoming and outgoing streams. The HiFi Glue itself consists of local and global sets of services. The glue and its relationship to the MDR and DSP are shown in Figure 3.

5.3.1. Local Services

The local HiFi Glue services perform actions that involve local decisions only.

---

[5] Our current implementation uses a combination of two versions of TelegraphCQ and the TinyDB system, as these have been previously developed by our group. The ease of incorporating other stream processors remains to be seen.

Logical Query Planner: The Logical Query Planner converts queries into a local query plan (the DSP Plan) and a set of queries to be run on child nodes (the Remote Query Set or RQS). We describe this process in more detail in Section 5.4.

DSP Manager: This module is responsible for starting, stopping, suspending, and modifying locally running DSP queries and streams based on input from the Query Planner. It also handles syntax translation from HiFi's internal query representation to the local DSP's query language, if necessary.

Resource Manager: This module's job is to adapt a node's behavior to unpredictable run-time conditions and perform functions such as prioritization and load-shedding.

Local View Manager: The Local View Manager provides a way to describe and manage the views that represent the data exposed by the node. It interacts with the MDR to export and revoke the current set of views active on this node. Additionally, it allows authorization, privacy, and other constraints to be specified for each view.

Archive Manager: This component manages the archiving of streams for the purposes of historical querying. Note that some DSPs support this functionality internally [13]. Furthermore, the Archive Manager may interact with other nodes to place data (both relations and streams) for efficient query processing.

Cache Manager: The Cache Manager snoops incoming data streams and determines what data to cache based on current workload. Additionally, it interacts with the Query Planner to enable query processing using cached data (i.e., materialized views).

Query Listener: The Query Listener listens for remote connections, parses incoming requests, and passes them on to the Query Planner.

Query Dispatcher: This component dispatches queries of the Remote Query Set to remote HiFi nodes.

Data Listener/Disseminator: These components handle incoming and outgoing data streams. Incoming streams from multiple sources may be merged into the same stream *en route* to the DSP. Outgoing streams may be split among multiple destinations. Additionally, these components handle pre- or post-processing of streams, such as encryption/decryption or format translation.

5.3.2. Global Services

Global HiFi services require non-local knowledge and interaction with other nodes in the system. In some cases, this portion of the glue need not physically reside on each node. Instead it can be viewed as a set of globally available services that perform actions on behalf of the requesting node.

Query Placement Service: This component determines the best node to start executing a query when received from a user. It does this by determining the scope and granularity of the query and consulting the MDR to determine the lowest common ancestor that can serve as the query root.

Physical Query Planner: Once the Logical Query Planner produces a set of plans involving child views, this module consults the MDR to determine the physical node location of these views. This is what enables fluid topologies in HiFi.

Control Manager: The Control Manager interacts with other HiFi nodes to perform overall system management. This includes system health monitoring and global and local startup/shutdown.

5.4. A Day in the Life of a Query

In this section, we illustrate the functionality of many of the architectural components by walking through the processing of a query.

The primary query planning mechanism on each node is called the *shared view infrastructure* (SVI), which is part of the Logical Query Planner. Each HiFi node is aware of a set of views (V_1, V_2,…, V_n) that describe all the data available to it (i.e., the views exported by all of its children). The SVI converts each view V_i to a succinct form composed of a set of sources and operators. The SVI then merges this view into its shared view representation in a manner identical to the way in which new queries are merged into a common shared plan in TelegraphCQ [12]. Thus, the SVI contains an agglomerated form of the individual views.

When a query Q enters the HiFi system, the Query Placement Service is consulted to dispatch the query to its root. Once a query arrives at its root, the following steps, coordinated by the Query Planner on each node, take place recursively to both plan and disseminate the query.

1. **SVI conversion:** Q is transformed into the same representation used for views in the SVI.
2. **Logical planning:** This plan is folded into the SVI in same manner in which views are added to produce the following:
    - **Remote Query Set (RQS)**: A set of queries (Q_{V1}, Q_{V2},…, Q_{Vi}), created by the Logical Query Planner, that represent the current query rewritten using the views of children. For each child node's view V_i that must provide data to this query, a corresponding query Q_{Vi} is created to be run on the corresponding child. Note that a query Q_{Vi} can be different from the view V_i as more operations can generally be pushed down into the provider of V_i.
    - **DSP Plan**: A local DSP query Q_{local} that operates over the input streams produced by the RQS.

3. **DSP setup:** The DSP Manager creates a new stream definition corresponding to each Q_{Vi} in the RQS and then adds Q_{local} on-the-fly to the DSP.
4. **Physical planning:** For each Q_{Vi} to be run on a child node, the Physical Query Planner produces the following:
   - **Where:** L_i - a location to run the query. If multiple child nodes export the same view, then the physical query planner chooses one of the nodes based on runtime conditions.
   - **What:** Q'_{Vi} **-** the actual query that has to be run. Note that Q'_{Vi} is a textual representation of Q_{Vi} in the syntax expected by the HiFi node L_i.
5. **Query dissemination:** Each Q'_{Vi} is sent to its appropriate location L_i where this same process takes place recursively.
6. **Data sourcing:** As results from the query Q'_{Vi} are received by the Data Listener, they are streamed into the DSP.
7. **Returning results:** As the local DSP produces results, the Data Disseminator directs them to the appropriate parent(s).

6. Prototype System

We have built an initial version of HiFi using the TelegraphCQ (TCQ) stream query processor and the TinyDB sensor database system. The goal of this prototype is to examine the feasibility of the uniform declarative framework and to derive a better understanding of the core components required for building high fan-in systems. Figure 4 depicts this initial deployment. It consists of a three-level hierarchy: receptors, initial processing, and core processing.

6.1. Receptors

The receptor level consists of sensor networks and RFID readers monitoring the physical world. For our sensornet system, we use TinyDB [28], which supports a SQL-esque interface for query processing. Our current prototype uses RFID readers to make up the other branch of the receptor level. Although the RFID reader does not export a SQL interface, we have built a simple adapter to interact with the device. Both of these systems are capable of some form of processing, ranging from aggressive in-network aggregation in TinyDB to simple buffering (i.e., windowing) in the RFID reader. HiFi exploits this functionality to clean the data as it samples it.

6.2. Initial Processing

The receptors feed their streams of partially cleaned data to the second level in our prototype hierarchy for initial processing. This level serves as the aggregation point for the receptors and consists of small computing devices capable of field deployment. For our system, we use Intel Stargates [16], small, single-board Linux-based compute devices built with Intel XScale processors. As shown in Figure 4, MoteServer and RFIDServer processes interact with the receptors and inject their streams into HiFi. For data processing, the Stargates run a scaled-down version of TelegraphCQ, capable of running simple continuous queries. Here the system performs additional cleaning and performs basic aggregation and correlation using queries similar to those shown in Section 3.2 before passing on the data. A Stargate along with an associated sensor network and RFID reader represent our *field deployable unit* (FDU) which monitors one *area*.

6.3. Core Processing

The Stargates feed their processed and aggregated streams to the root of our hierarchy, a full-fledged server running TelegraphCQ. This node runs queries that correlate streams across all devices, for example, "find the maximum (sound/number_of_tags) quotient across all areas."

6.4. Experiences

We have deployed this prototype as described with several FDUs and a streaming visualization interface for demonstration at the 2004 VLDB conference [15]. In many ways this experience influenced the design presented above. We briefly discuss some of these experiences and the lessons we learned from them.

There were many basic problems arising from the inherent complexity of a high fan-in system. Each new device incorporated into the system brought with it its own implementation challenges. We discovered many small bugs in different parts of the system as we moved to each new platform. These challenges provide a strong argument for a general-purpose data management platform such as HiFi to remove this source of complexity when deploying high fan-in systems.

Although our deployment had only three levels in its hierarchy, we discovered that there was severe data lag, from the time when a receptor read a data value to the time when that value was reflected in the output. This stemmed from the fact that hierarchical, window-based query processing, naively implemented, has inherent delays. If a query's windowing specifications (i.e., its *range* and *slide* parameters) are disseminated unchanged through the hierarchy, a lag on the order of the slide parameter is introduced for each level. Thus, the system must be careful in handling the window clause as a query is propagated down the hierarchy. We are currently developing techniques to address this issue.

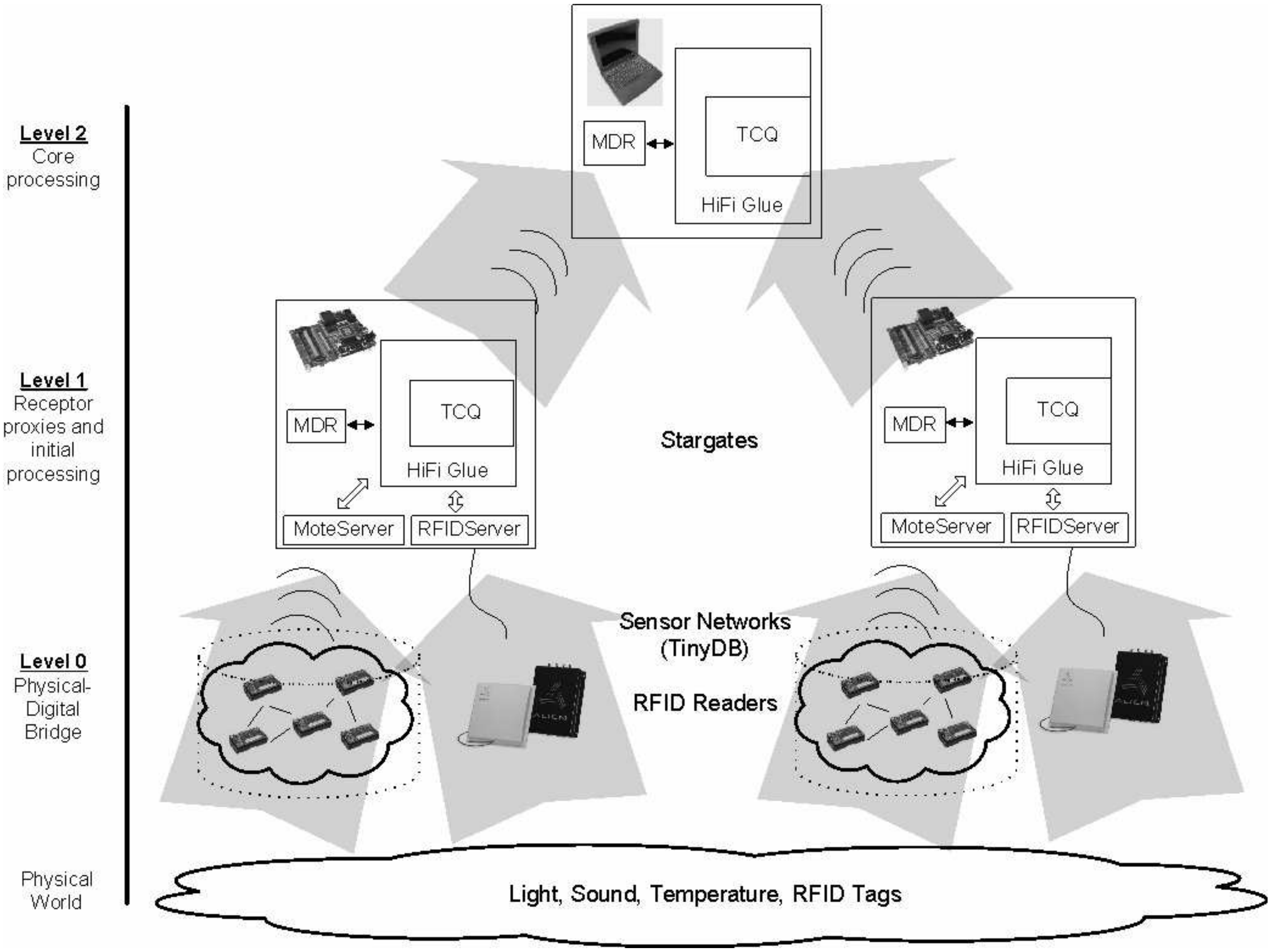

Figure 4 – The initial HiFi prototype [15]

Our implementation of CSAVA provided many interesting lessons. As development and testing progressed, we discovered that the RFID data produced by the readers were highly unreliable. By applying CSAVA processing, we were able to clean up this data to a certain extent. Additionally, this effort validated our uniform declarative framework, as CSAVA deployment took relatively little development time.

Finally, deploying this system reinforced our belief that system management is very important. Our deployment was relatively small in scope, yet it had five different platforms running four different data processing systems across more than 20 devices. Without management tools for start up, shutdown, and status, the system would have been largely unusable. Additionally, we discovered that compartmentalized design of each node provides a benefit in that faults were isolated to the component that failed. We were able to dramatically increase the uptime of the system through strict compartmentalization of each component in the system.

7. Related Work

As has been discussed in the previous sections, HiFi builds on a large body of related work. In addition to work already discussed about the individual components of HiFi (e.g., view-based query rewrite), there is previous work relating to both the architecture of high fan-in systems in general and to the design of HiFi in particular.

7.1. Hierarchical and Receptor-based Systems

There are a variety of projects aimed at managing and querying the data produced by receptors, both physical and virtual. These projects have assumed topologies similar to the high fan-in approach described here, although there are significant differences.

IrisNet[17] uses a two level hierarchy consisting of receptors feeding into a core composed of a set of nodes running a distributed database. Queries are posed in XQuery over a hierarchical schema which represents both the node organization and data organization. Thus, queries contain full information to enable the query to be routed to the lowest common ancestor necessary to answer that query. The focus is on ease of service deployment and scalability. Although IrisNet is organized in a hierarchy, it does not address hierarchical aggregation or successive processing of queries.

Astrolabe [43] is designed for distributed system monitoring and data mining for system management. It organizes its nodes in a hierarchical manner (termed *zones*) with a primary focus on aggregation to enable system scalability. While not explicitly dealing with streams, it does handle rapid updates to the underlying data and re-computes aggregates on-the-fly. It does not address windowing semantics. Astrolabe is designed to run on a relatively homogenous system and doesn't take into account differing system capabilities.

The MIT Auto-ID center defines a set of specifications on how to interact with RFID data, including Savant [31]. They address a similar hierarchical framework with multiple Savants talking to each other. The also define some of the same types of data processing stages we discuss in our CSAVA example. However, each stage in their processing involves a different data model and different protocol for interacting with the data.

The Hourglass project [40] from Harvard is developing a data collection network (DCN) for accessing sensor-based data. Their infrastructure consists of an overlay network of wired nodes collecting data from various sensor networks. They generalize system components into *producers*, *consumers*, and *services* and focus on how best to establish and maintain *circuits* in the overlay network.

The D-Stampede project [3][35] at Georgia Tech provides a programming system for managing what they term an "Octopus" hardware configuration, with a wide range of receptors feeding into a cluster for further processing. Their goal is to provide an API to support high performance application development for a heterogeneous (both hardware and software) environment. They focus on providing an application development environment and not on data management.

7.2. Data Stream Processing

As we have discussed previously, HiFi draws heavily from the large body of recent work on single site data stream processing. Projects in this area include TelegraphCQ [12], STREAM[30], Aurora [1][10], and NiagaraCQ [11]. To date, there has been less work on distributed stream processing.

The Aurora Project has branched into two separate efforts to extend stream processing to a distributed environment. Aurora* [14] is designed for a single administrative domain and addresses QoS and dynamic operator repartitioning and movement to achieve load-balancing and fault-tolerance. The Medusa System [14] arranges single site Aurora data stream processors in a loosely federated network mediated by agoric principles to enable spanning of organizational boundaries and load balancing. This work differs from HiFi that it has focused on distributing stream processing for load balancing and high availability. In contrast, HiFi is focused on identifying and addressing the problems that arise in systems that naturally assume a high fan-in topology.

More recently, Ahmad and Cetintemel have reported on an in-depth study of operator placement in a distributed stream processing system [5]. This work analyzes multiple algorithms and exposes the trade-off between bandwidth usage and answer latency.

7.3. Distributed Data Management Systems

More traditional database research has focused on distributed data management in the form of both tightly and loosely coupled distributed databases as well as federated databases. Relevant efforts in this area include Mariposa[42], Information Manifold [25], and Tukwila[23].

8. Conclusions

In this paper, we have introduced the notion of high fan-in systems, an emerging information systems architecture that leverages advances in data acquisition and sensor technologies to enable disparate, widely distributed organizations to continuously monitor, manage, and optimize their operations. The technology required to support high fan-in systems builds on previous work in federated data management, data stream processing, sensor network query processing, and distributed data management; however, the unique architectural, application, and environmental considerations that arise in such systems raises a wealth of new and interesting research questions.

We described our initial design ideas and outlined currently open issues in the development of HiFi, a high fan-in infrastructure currently being implemented at UC Berkeley. We have built an initial prototype of HiFi using the TelegraphCQ and TinyDB code bases, and have successfully demonstrated the usefulness of stream-oriented query processing for correlating, aggregating, and visualizing readings from sensor motes and RFID readers. This paper represents a current snapshot of our development and identifies areas of future research. Of course, as the project develops, we anticipate that both our design and our research agenda will evolve as new issues and opportunities arise.

References

[1] Abadi, D. et al., Aurora: A New Model and Architecture for Data Stream Management. In VLDB Journal, (August 2003).

[2] Abadi, D., Lindner, W., Madden, S., Schuler, J., An Integration Framework for Sensor Networks and Data Stream Management Systems. VLDB 2004.

[3] Adhikari, S., Paul, A., and Ramachandran, U., D-Stampede: distributed programming system for ubiquitous computing. ICDCS 2002.

[4] Afrati, F. and Chirkova, R., Selecting and Using Views to Compute Aggregate Queries, (unpublished manuscript, 12/08/2003).

[5] Ahmad, Y. and Cetintemel, U. Network-Aware Query Processing for Stream-based Applications. In VLDB (2004).

[6] Acharya, S., Alonso, R., Franklin, M., and Zdonik, S., Broadcast Disks: Data Management for Asymmetric Communications Environments. In SIGMOD (1995).

[7] Balazinska, M., Balakrishnan, H., and Stonebraker, M., Load Management and High Availability in the Medusa Distributed Stream Processing System. In SIGMOD (2004).

[8] Bonnet, P., Gehrke, J., and Seshadri, P., Towards Sensor Databases. In MDM (2001).

[9] Bonnet, P., and Seshadri, P., Device Database Systems. In ICDE (2000).

[10] Carney, D. et al., Monitoring Streams - A New Class of Data Management Applications. In VLDB (2002).

[11] Chen, J., DeWitt, D., Tian, F., and Wang, Y., NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In SIGMOD (2000).

[12] Chandrasekaran, S. et al., TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In CIDR (2003).

[13] Chandrasekaran, S., and Franklin, M., Remembrance of Streams Past: Overload-Sensitive Management of Archived Streams, In VLDB(2004).

[14] Cherniack, M. et al., Scalable Distributed Stream Processing, In CIDR (2003).

[15] Cooper, O. et al., HiFi, A Unified Architecture for High Fan-In Systems (System Demonstration), In VLDB 2004.

[16] Crossbow. The Stargate single board computer. http://www.xbow.com/Products/XScale.htm.

[17] Deshpande, A., Nath, S., Gibbons, P. B., Seshan, S. IrisNet: Internet-scale resource-intensive sensor services. ACM SIGMOD 2003.

[18] Duschka, O. M., and Genesereth, M. R., Answering Recursive Queries Using Views. In PODS (1997).

[19] EPCGlobal, EPCGlobal Homepage, *http://www.epcglobalinc.org*.

[20] Faradjian, A., Gehrke, J. E., Bonnet, P. GADT: A probability space ADT for representing and querying the physical world. In Proceedings of ICDE 2002 (2002).

[21] Hellerstein, J. et al., Adaptive Query Processing: Technology in Evolution, *IEEE Data Engineering Bulletin,* June 2000.

[22] Harren, M. et al., “Complex Queries in DHT-Based Peer-to-Peer Networks”. In IPTPS (2002).

[23] Ives, Z. G., Florescu, D., Friedman, M., Levy, A., Weld, D. S., An Adaptive Query Execution System for Data Integration. In SIGMOD (1999).

[24] Kossmann, D., Franklin, M., and Drash, G., Cache Investment: Integrating Query Optimzation and Distributed Data Placement, ACM TODS Vol. 25, No. 4, Dec. 2000.

[25] Kirk, T., Levy, A., Sagiv, J., Srivastava D. The information manifold. Technical report, AT&T Bell Laboratories, 1995.

[26] Lerner, A. et al., AQuery: Query Language for Ordered Data, Optimization Techniques, and Experiments, In VLDB 2003.

[27] Levy, A., Rajaraman, A., and Ordille, J. J., Querying Heterogenous information sources using source descriptions. In VLDB (1996).

[28] Madden, S., Franklin, M., Hellerstein, J., Hong, W., A Tiny Aggregation Service for *ad hoc* Sensor Networks, In OSDI (2002).

[29] Madden, S., and Franklin, M., Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data, In ICDE (2002).

[30] Motwani, R. et al., Query Processing, Resource Management and Approximation in a Data Stream Management System, In CIDR (2003).

[31] Oat Systems and MIT Auto-ID Center. The Savant. Technical Report MIT-AUTOID-TM-003, MIT Auto-ID Center, May 2002.

[32] Pottinger, R., and Levy, A., A Scalable Algorithm for Answering Queries Using Views, In VLDB (2000).

[33] Qian., X., Query Folding. In ICDE (1996).

[34] Qin, S. J., "Neural networks for intelligent sensors and control --- Practical issues and some solutions," In: O. Omidvar and D.L. Elliott (Ed.), Neural Systems for Control, Academic Press, chapter 8 (1997).

[35] Ramachandran, U. et al., Stampede: A Cluster Programming Middleware for Interactive Stream-oriented Applications. IEEE Trans. on Parallel and Distributed Systems, Nov. (2003).

[36] Ramakrishnan, R. et al., SRQL: Sorted Relational Query Language, In Statistical and Scientific Database Management 1998.

[37] Reiss, F., and Hellerstein, J. , Data Triage: An Adaptive Architecture for Load Shedding in TelegraphCQ, ITB-TR-04-004, Intel Research, February (2004).

[38] Rizvi, S., Mendelzon, A., Sudarshan, S., and Roy, P., Extending Query Rewriting Techniques for Fine-Grained Access Control, SIGMOD (2004).

[39] Sarbanes-Oxley Act. http://www.sarbanes-oxley.com

[40] Shneidman, J. et al., Hourglass: An Infrastructure for Connecting Sensor Networks and Applications. Harvard Technical Report TR-21-04

[41] Srivastava, D., Dar, S., Jagadish, H. V., Levy, A. Y., Answering Queries with Aggregation Using Views. VLDB 1996.

[42] Stonebraker, M., Aoki, P. A., Litwin, W., Pfeffer, A., Sah, A., Sidell, J., Staelin, C., and Andrew Yu. Mariposa: A Wide-Area Distributed Database System. VLDB Journal, 1996.

[43] van Renesse, R., Birman, K. P., and Vogels, W., Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. ACM TOCS, 2003.