Towards a Pervasive Grid[*]

Vipul Hingne, Anupam Joshi, Tim Finin, Hillol Kargupta
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
Baltimore, MD 21250
Email: {vipul1,joshi,finin,hillol}@cs.umbc.edu

Elias Houstis
Department of Computer Sciences
Purdue University
West Lafayette,IN 47907
Email: enh@cs.purdue.edu

Abstract

The increase in the use of mobile & embedded devices, coupled with ad-hoc, short range wireless networking is enabling pervasive *computing. This pervasive computing environment and the wired Grid infrastructure can be combined to make the Computation and Information Grid truly pervasive. This paper identifies some of the interesting research issues and challenges in creating such a pervasive Grid, and describes some preliminary work we have done to that end. We propose a runtime environment for the Pervasive Grid that utilizes a multi agent framework, and provides for discovery of services being offered by sensors, embedded and mobile devices, and their composition. The computation in this environment needs to be dynamically partitioned between the traditional Grid and elements that constitute the pervasive environment, like sensors, with limited computing and communication capabilities.*

1 Introduction

The evolving information infrastructure has significantly impacted many facets of society over the past decade, and promises to play an increasingly important role in our lives. An important component of this will be the continuous interaction between individuals, embedded devices and sensors, and the wired information infrastructure that the wireless networks will engender. This vision has been described as *pervasive computing* and is enabled by the rapidly shrinking form factor/cost of computers for a given computational power, and significant new advances in the creation of a variety of wireless networks. Contrasted with the high bandwidth information superhighways with which the computational grid has mostly concerned itself, wireless networks will serve as the country roads which provide "ubiquity of access". The vision that drives our research is one of providing seamless access to networked computational resources (a.k.a "The Grid") from wirelessly networked walk stations (laptop/palmtop/wearable devices), embedded sensors and controllers, and nanosensors . We are investigating an *agent oriented approach* towards enabling this vision, which is a component of the idea of scalability of the information infrastructure that a recent presidential advisory panel recommended. More specifically (and in the short term), the work we propose here will create a framework and agent oriented middle-ware to realize the Pervasive Grid. In ongoing work, we are

- Developing an agent oriented software architecture appropriate for UPSEs that will promote reuse of legacy scientific computing codes and handle their evolution.
- Developing techniques for multi agent systems to dynamically adapt to their changing environment. These techniques will create a framework where software components/agents advertise their capabilities, discover other agents, and negotiate with other agents about appropriate mediating interfaces or performance commitments etc.
- Developing techniques allowing ubiquitous access to the networked scientific computation infrastructure.

Networked Scientific Computing (NSC) systems developed over the last few years have enabled us to attack large multidisciplinary scientific problems. They allow us to use the high performance communication infrastructure (vBNS, NGI, Internet II etc.) to view heterogeneous networked hardware (from the ASCI terraflop machines to workstations) and (legacy) software (e.g. specialized solvers, databases of material properties, performance measuring systems) resources as an abstract, single "meta computer"[15]. This scenario is typically tailored towards a scientist or engineer at a desktop machine attacking a known problem. The emergence of pervasive computing

[*]This work was supported in part by NSF Awards 0203958, 9875433, and 0070802

will not just create a new infrastructure that the Grid must accommodate, but create a new class of applications as well.

Consider a real time environment to monitor the health effects of environmental toxins or disease pathogens on humans. There are significant advances being made today in biochemical engineering to create extremely low cost sensors for various toxins[18] that could constantly monitor the environment and generate data streams over wireless networks. It is not unreasonable to assume that similar sensors could be developed to detect disease causing pathogens. In addition, most state health/environmental agencies and the federal government entities such as CDC and EPA have mobile labs and response units that can test for the presence of pathogens or dangerous chemicals. The mobile units will have handheld devices with wireless connections on which to send the data and/or their analysis. In addition, each hospital today generates reports on admissions and discharges, and often reports that to various monitoring agencies. Given these disparate data streams, one could analyze them to see if correlates can be found, alerting experts to potential cause-effect relations (Pfiesteria found in Chesapeake Bay and hospitals report many people with upset stomach who had seafood recently), potential epidemiological events (field units report dead infected birds and elderly patients check in with viral fever symptoms, indicating tests needed for west nile virus and preventive spraying), and more pertinent in present times, low grade chemical and biological attacks (sensors detect particular toxins, mobile units find contaminated sites, hospitals show people who work at or near the sites being admitted with unexplained symptoms). At present, much of this analysis is done "post facto" – experts hypothesize on possible causes of ailments, then gather the data from disparate sources to confirm their hypotheses. Clearly, a more proactive environment which could mine these diverse data streams to detect emergent patters would be extremely useful.

As another example, consider a real-time environment for monitoring and commanding a defense operation. It may involve a central command and control station, airborne vehicles and sensors (e.g. AWACS, drones), ground-based wireless integrated network sensors, satellite data, and war fighters on the ground. Defense applications like this often require emergent situation awareness, discovery of anomalous patterns, and performance assessment. This in turn requires analysis of massive amount of data available through sensor data streams, databases containing related information such as intelligence reports, target tracks, road networks, as well as databases with information about the weather, site models, and enemy deployment. The war fighter on the ground may be interested in finding out enemy capabilities in his neighborhood; the control team in the airborne AWACS may be interested in monitoring the major active weapons system and detection of any anomaly. On the other hand the main mission control may want to query the data network for evaluating the overall performance or simulating the potential effect of their actions.

Analysis of this kind of complex data sets for different purposes requested by different groups of users may involve a whole range of different techniques like clustering, predictive scoring, solving Navier-Stokes equations for applying numerical models. This may involve light-weight *in situ* analysis of data in the sensor nodes or streaming of data to high-end number crunching machines for running large simulations. We should note here that the obvious solution, of gathering all the data at a central place, is not likely to work. The sensors, if treated as dumb data sources only, can generate huge data streams that will be beyond the capacity of the wireless connections, and will drain battery power at sensors which can be in short supply. Often the sensing elements or the field units will need to minimize the traffic they generate so as to avoid detection and potential destruction. In that case we will need inherently distributed techniques to analyze data in such a way that resource usage (network communication load, sensor network power consumption) is minimized on the constrained nodes.

We note that the two scenarios painted above, far from being unique, are actually representative in fields as far apart as process monitoring & control, and network-based intrusion detection. The runtime system needed to enable such scenario would need to posses the following characteristics:

- It should be able to handle the transport level problems caused by low bandwidth, high latency, frequent disconnections and network topology changes.
- It should be able to operate on a variety of devices with different resource constraints
- It should automatically figure out the components needed for a task and discover them on the network
- It should be able to compose the discovered components to dynamically create the required application
- It should be able to dynamically partition the computation across the various computational elements constrained by device resources, data and code locations, and network bandwidth/connectivity.

Notice that some of these issues do not arise in the traditional Grid Computing work that has been done for fixed networks – such as extreme variability of device resource constraints, dynamic network topologies etc. Others are similar to the ones that the Grid Computing community has dealt with (recommending components, distributing computations), but have to operate under significant new constraints associated with pervasive computing and so will require the development of new approaches. We propose the use of *multi agent* systems to enable the Pervasive Grid.

In the next section we describe the Multi agent Framework that underlies our Runtime Environment. Then we discuss our approach for Service Discovery, Composition and Management. Finally we describe our ongoing work on Dynamic Partition of Computation in sensor networks.

2 Multi agent Framework

The DARPA Knowledge Sharing Effort [23] has given rise to a model for developing intelligent cooperating agents based on ontologies, content languages and agent communication languages. Although this approach and its associated components is being adopted as a standard for agent communication by both the research community and the industry, it requires significant extensions to support the inter agent interactions and negotiations which our system will demand. We will attack a number of key problems, developing theoretically sound solutions for each, building those solutions into our evolving software infrastructure, and experimenting with the results in the wireless communication environment. We will design our system to scale across the problem complexities and reaction time needed for applications, ranging from simple but fast reactive actions of an agents doing network bandwidth measurements to the more complex, slower cooperation needed for cooperative analysis of heterogeneous data streams across wireless networks.

Agent Development Framework
The underlying substrate of the system will be provided by further development of our Ronin Agent Framework [6, 7]. At present, this is a simple Jini-based agent development framework that is designed to aid in the development of next generation smart distributed mobile systems. In the Ronin Agent Framework we introduce a hybrid architecture, a composition of agent-oriented and service-oriented approaches. Ronin contains a number of features that distinguish it from traditional multi agent frameworks and will make it especially suitable for the mobile domain. These include an ACL and network protocol independent communication infrastructure. Ronin has the notion of service discovery (agent discovery) built into the architecture – this allows us to integrate our new service discovery protocols into it. We will integrate the enhanced Ronin functionalities into one of the existing FIPA compliant Agent Development Platforms such as JADE or AGILE.

The design goal of Ronin is to define an open framework that specifies the infrastructure requirement and the interface guideline for the interaction and communication between agent-oriented components. It models services as agents. Each service consists of two parts: an Agent Deputy and an Agent. An Agent Deputy acts as a front-end interface for the other agents in the system to communicate with the Ronin Agent it represents. Ronin Agent is a service that can be realized as software or hardware. Ronin does not define how an Agent Deputy should communicate with a Ronin Agent. However, it does define the interface for the communication between the Ronin Agent and Agent Deputy. Specifically, each Agent Deputy must implement a `deliver` method. This delivery abstraction means that depending on their connectivity and network QoS, agents can deploy deputies that will provide features of transcoding or disconnection management that we will develop as a part of this research. The messages that are interchanged between Ronin Agents are embedded within `Envelope` objects during the delivery process. This meta-level approach allows Ronin Agents to interchange messages with arbitrary content message types under a uniform communication infrastructure [30]. Within each `Envelope` object, the type of content message and the ontology identifier of the content message are also stored.

There is a set of attributes associated with each Ronin Agent. These attributes can be further divided into two subsets. The first set of attributes, Agent Attributes, define the generic functionality of an agent in domain independent fashion. For example, an agent could be a broker, or a service provider. Ronin framework defines the types and the semantics of Agent Attributes. The second set of attributes, Agent Domain Attributes, define the domain specific functionality of an agent. For example, in a financial domain, an agent could be a stock quote server. The framework neither defines the Domain Attribute types nor their semantics. While domain attributes will allow us to create agents that understand a domain specific ontology, agent attributes provide a common base from which interaction amongst agents from heterogeneous domains can be bootstrapped.

3 Service Discovery, Composition, and Management

In order for an entity to cooperate with others in its vicinity or use the services available on the fixed grid, it needs to discover them. This problem of "service discovery" has recently been explored elsewhere as well in the context of distributed systems. We use the term service broadly here – it could be a computational component which executes, data/information, or even a CPU cycles / storage capacity that one entity is willing to provide to the other. Thus there is a need to develop mechanisms which allow for components to describe themselves (at a semantic level) and their "requirements", as well as for other components to locate them.

State of the art systems such as Jini[1], Salutation[26], UPnP[26], IETF's Service Location Protocol[12], E-Speak[13], Ninja[8], and most recently, UDDI[9] provide for networked entities to advertise their functionality. However, these systems are either tied to a language (Java/Jini), or describe services entirely in syntactic terms as interface descriptions. This not only limits interoperability, but forces a client to know a-priori how to describe a service it needs in terms of an interface. Moreover, they return "exact" matches and can only handle equality constraints. This leads to a loss of expressive power in the component description. For example, the Jini discovery and lookup protocols are sufficient for service clients to find a service that implements the method `printIt()`. However, they are not sufficient for clients to find a printer service that has the shortest print queue, that is geographically the closest, or that will print in color but only within a prespecified cost constraint. When we look at service discovery in short range ad-hoc mobile environments like Bluetooth, the picture is even worse. Bluetooth SDP relies on unique 128 bit UUIDs to describe and match services. This is clearly inadequate.

We are investigating the creation of efficient broker agents to discover services at a semantic level. To the extent possible, we leverage emerging industrial standards such as UDDI, BPML and WSDL/WSFL. UDDI's present highly centralized model is not appropriate for our scenario, but more recent developments in that effort seem to indicate that a distributed set of brokers could be created. In order to describe services, we use semantic web languages. At present we use DAML/DAML-S (DARPA Agent Markup Language, `http://www.daml.org/`), the semantic language developed by a joint US/EU effort over the past few years. We plan to switch to OWL, a DAML based language being standardized by the W3C at present. Each entity/component of the system will be able to describe itself using DAML. Components register their capabilities (what services they can provide) and constraints/requirements (what software/hardware they need to run, how much is the cost to run them, what interfaces they provide) using DAML. The matching of a request to services is semantic and uses the DAML descriptions. This matching is fuzzy, and often recommends a ranked list of matches. Our initial work towards building such a system is described in [19, 4, 2]. We are currently enhancing the matching process to fully utilize the logic description that DAML+OIL provides.

Given an efficient semantic level discovery infrastructure, the next task is to use it to compose services and components. For instance, a particular analysis technique[17] for streams tries to create ensembles of decision trees from the data stream and then combine them. First the system needs to figure out that this task has several components – generating decision trees, computing their Fourier spectra, choosing the dominant components, and combining them to create a single tree. For task categories that are well understood *a-priori*, this can be done by hard coding specific decompositions. However, in the more general case, this requires the use of a planner[11]. We feel that existing planning techniques are adequate for our purposes, and we will eventually integrate a planner like SPIE-2[29] into our system.

The second part of the composition problem is actually putting together the composite task in a mobile or mobile-wired environment. Given a certain ordering of several sub tasks that may be executed to derive the result of a complex request, the problem is how these heterogeneous tasks can be integrated and executed in environments where there is a combination of resource-rich and resource-poor devices interconnected to each other by thin or thick communication channels. The problem can be tackled by using centralized broker-based architectures [22, 3, 10] for service composition in purely wired environments. However, in pervasive grid systems where the computation platforms range from high end super computing workstations to low-end minute nano sensors, centralized architectures are often not the most appropriate. We highlight some of the important research issues associated with service composition in such environments.

Service coordination and management is one important aspect for service composition. Every service composition platform must have some entity coordinating the different services involved in the composition. Most service composition platforms follow a centralized architecture to coordinate and manage the execution of a composite service. The efficiency and design of the platform depends on the way the central entity or manager handles the different services. The problem of coordination and management becomes difficult when the entities are distributed across the network and when some services are resident in a relatively stable fixed 'wired' environment and some services are more mobile and present in the vicinity of a client. The composition architecture needs to ensure that the composite service is tolerant to failures, available and efficient to a reasonable extent. The composition manager should also be able to efficiently manage the network resources and apply certain optimization criteria to reduce the cost of the composite service . Service composition becomes difficult in a resource-variant environment consisting of heterogeneous devices with varying capabilities. A good service composition architecture in a mobile/ubiquitous environment should ideally be able to use the maximum amount of resources it can obtain from its vicinity to compose a complex query. Such resource utilization calls for a distributed way of implementing a composition architecture. Services themselves may be resident on mobile devices in an ad-hoc environment. For example, there might be sensor services willing to provide information about the pollution/germ indices in the air in its vicinity. Service composition should be able to take advantage of different short-lived services which stay in the vicinity for a finite amount of time and then disappear. A distributed service composition platform should follow the mobility pattern of a set of services. There may be different ways to carry out service composition of requests depending on the frequency of requests. We might want to pro-actively compute some generic information about services required to execute a query which is requested with a high frequency. The other approach is to re-actively integrate and execute services to derive the result of a query. In this research we aim to investigate various means of achieving efficient resource/computation usage in the form of service composition in ubiquitous environments.

We need different services following different information exchange mechanisms to operate together to realize a heterogeneous service composition platform. Examples of such mechanisms include services that follow the message-passing paradigm to communicate with clients, services that follow the remote method invocation mechanism like SOAP[25] or agent-based services that follow a certain agent language. A good service composition platform should be able to communicate with all the different services and utilize them appropriately for any composition with other services not following the same type of information exchange paradigm.

Fault tolerance and scalability are two most important aspects for efficient service composition and information retrieval. In a conglomeration of super computing grids and low-end wireless grids, there are ample chances of link and resource failures. If a network service breaks down, the architecture should be able to detect this and resort to fault control mechanisms to ensure that the processing of a complex query is not hindered. The composition platform should degrade gracefully as more and more services become unavailable. Scalability is also an important aspect in this domain. Composition architectures should scale with the increasing number of services in smartdust type environments.

The architecture should have the flexibility to adapt itself to the changing environment in the smart space layer of the ubiquitous grid. The world of services can change rapidly especially in the lower layers of a grid computing system where there are *m* heterogeneous mobile devices. Services may be coming up and going down frequently in those environments. There are also examples of long-standing services (e.g. weather forecast, problem solvers) in fixed grid environments. With the induction of mobile devices, the chances of non-persistent services being available for sometime are increasing. A good composition platform should be able to adapt its composition by taking maximum advantage of the currently available services. This increases the chances of availability of the composite service in a dynamically changing environment.

Apart from just being able to integrate and execute different simple services and hence achieve better utilization of networked resources, execution also needs to be done in an efficient manner. This is particularly relevant in a mobile or mobile-wired environment where the same service plan may be executed in different manners to optimize parameters like bandwidth, data transfer. Research needs to be done in this field to enable efficient and fast resource utilization. Our preliminary work in creating such a composition architecture in described in [5]. We have built a system that can do pure reactive composition. The implementation is over a combination of notebook and PocketPC devices, and uses Bluetooth as well as 802.11. We have also extended GloMoSim [31] to simulate such dynamic composition, and are carrying out simulations.

4 Dynamic Partition of Computation in Pervasive Environments

There has been a lot of work in partitioning of computation in the Grid computing community. However, as pointed out earlier, different issues arise when the sources of data are tiny sensors, the network topology is dynamic, the connections are wireless, and there is vast difference between the computing abilities of the devices making up the grid. Let us look at a scenario which illustrates many of these issues.

Scenario

Consider a building with temperature sensors embedded at various locations in the building. The sensors can communicate with neighboring sensors and with a base-station located nearby. They generate streams of temperature data. The Grid Infrastructure, capable of doing heavy computation is accessible from the basestation. Suppose the building is on fire. Fire fighters with handheld devices arrive, and want to query the sensor network in the building to plan their response. The queries could be as simple as finding the temperature at a particular spot, or involve a simple computation such as the average temperature in some room. Such queries (or computations) can be done in-situ in some combination of the sensors and the basestation. Other queries could be as complex as finding the temperature distribution inside the building. To answer this query, a 3D partial differential equation needs to be set up, grid points populated by data from the sensors and static data about building material and boundary conditions, and then solved. It is simply not feasible to perform the computation for solving such a query inside the network. One way would be to transfer the data from the sensors to the grid and perform the computation in the grid. Even in this case, one might want to perform some kind of optimizations to speed up the response time. For instance, depending upon the accuracy of results required, instead of sending each sensor reading to the grid, one might only send the average reading from a region(the size of the region depending on the level of accuracy needed).

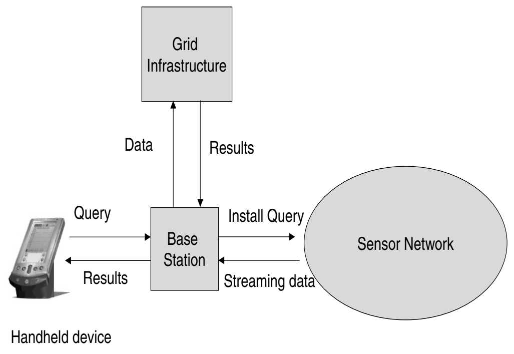

Figure 1. General Scenario

The problem that we intend to solve is to dynamically partition the computation needed for the execution of the query. Some obvious possibilities for this are

- The data is moved to the resources on the grid, which do the computation
- The computation is done in the sensor network and only the result is provided to the base station/handheld devices
- The data is delivered to the base station/PDA, which perform the computation

Some queries may involve performing a lot of computation, which may take a long time in the sensor network. Such queries are best solved by the first approach above. Some very frequent queries may require less computation, but the amount of data transfer required may drain the energy of the sensors. The second approach would work best in this case. Some queries which fall between the earlier two may be best solved by the last approach above. Some queries may need combination of the approaches above. It is important to make this decision whenever a query is submitted.

To be able to dynamically partition the computation some estimates would be needed. It is essential to know the amount of computation required for a particular query. Another important parameter is the amount of data transfer required for evaluation of the query. In sensor networks, preserving the energy of the sensors is of prime importance. So estimates of energy consumption of sensors to evaluate a query with each of the above approach are desirable. For real-time queries, the turn around time is crucial. Hence estimate of the response time of the query in each of the above approach is needed.A lot of factors would affect the estimates required above. All networks may not be of the same size, so the number of sensors in the network would wary. Different networks would have different network topology. The data routing technique used in the network would not be the same for all networks. A particular network may use flooding technique to route data, while another may use gossiping. Different sensors may generate data with different rates.

To ease the process of making the various estimates described earlier, we have divided the possible queries into four different types.

- Simple Queries: Queries targeted at a particular sensor, would fall under this category. E.g. “Return temperature at Sensor # 10”
- Aggregate Queries: Queries which involve aggregate functions like Max, Min, Avg, Sum, etc. E.g. “Return Average Temperature in room # 210”
- Complex Queries: Queries which involve performing computation over data from sensors to return the result. E.g. “Find Temperature Distribution in room #210”
- Continuous/Windowed Queries: Any query which is continuous in nature. E.g. “Return temperature at Sensor #10 every 10 seconds”

The query format we use is presented below:

```
SELECT {func(), attrs} from sensors
WHERE { selPreds }
COST { cost limitation }
EPOCH DURATION i
```

The query format is similar to the one used by Madden et. al in TAG[21]. However we allow for any arbitrary function to be specified in the SELECT clause. We have also introduced the COST clause to specify the cost within which the function is to be evaluated. Cost could be in terms of sensor energy, response time or accuracy of the result. The EPOCH clause specifies the interval between two consecutive results for continuous queries.

Different solution models can be used to gather data and perform the computation required to answer a query. In a simple model, all sensors would send their data to the base station. The base station would then perform the computation over the data. Cluster based models can enable the computation to be carried out in the sensor network. Sensors are divided into clusters and each cluster has a cluster head. Cluster heads aggregate information from the sensors in individual clusters and send it to the base station. Another way to perform in-network aggregation is to use aggregation trees[16]. Data centric routing techniques can be used to form aggregation trees in sensor networks. Data would be routed and aggregated through the aggregation trees. Most importantly, the grid can be used to perform the computation. The data would be transferred to the grid through the base station. The computation would be done in the grid and results would be returned to the base station. For a given query, one or a combination of the models above can be used to perform the computation.

In Pythia[14], we have shown that Artificial Intelligence techniques can be successfully applied to Problem Solving Environments. We choose a similar approach here to dynamically partition the computation required to evaluate queries. We propose to conduct simulations on these query types to generate data for amount of computation, data transfer, energy consumption, and response time for various approaches. Standard machine learning techniques would be used on the data to select the right approach for a given query. The system will be made adaptive by comparing the estimates of energy consumption and response time with the actual values of energy consumption and response time during the execution of the query and the results would be incorporated into the learning technique.

The system comprises of three major components: Query Processor, Decision Maker and Simulator for sensor network. Query processor analyzes the query and categorizes it into one of the types mentioned above. Decision maker would decide the solution model to use based on type of query, historic data and known features of the network at hand. The simulator simulates the solution model for the query and returns the results.

To the best of our knowledge, this is the first work which utilizes the the computational abilities of the grid to answer queries on streaming data from sensor networks. There have been efforts in the sensor database community that look into efficient query processing in sensor networks, but none utilize the grid.

Johannes Gehrke and colleagues from Cornell University present a model for sensor databases in their Cougar Sensor database project[24].Sensor databases consist of stored data; regarding the sensors, and the sensor data. The authors represent the stored data as relations and sensor data as time series. The emphasis is on modeling long running queries. The sensor time series is based on the sequence model introduced by Seshadri et al[27]. Sensor queries involve relations and sequences. Relations are manipulated using relational operators and sequences are manipulated using sequence operators. In COUGAR, sensors are modeled using ADTs and the signal processing functions of the sensors are modeled by ADT functions which return the sensor data. Long running queries are formulated in modified SQL. Signal processing functions are modeled as scalar functions which disallow the use of queries with explicit time constraints like aggregates. Cougar, however supports only a limited type of queries.

Micahel Franklin and colleagues have proposed a query plan data structure called Fjords(Framework in Java for Operators on Remote Data Streams)[20] to allow queries combining push-based sensor sources and conventional pull-based sources. The system provides non-blocking and windowed operators over streaming data. They also propose sensor proxies which act as mediators between query processing environment and the physical sensors.

Samuel Madden et. al from UC Berkeley propose aggregation of data in sensor networks as part of their TinyDB project TAG (Tiny Aggregation)[21]. They push declarative queries into network. They impose a hierarchical routing tree onto the network. In TAG, Madden et. al show that performing the computation for certain type of aggregate queries inside the sensor network result in saving the energy of the sensors and thus lengthen the lifetime of the sensor network. They also suggest further optimizations like channel sharing which result in further saving of sensor energy. Their approach, however, is designed primarily for aggregate type queries. They do not support complex functions in queries.

Viglas et. al from University of Wisconsin- Madison present Rate-Based query optimization for streaming information sources [28]. They argue that cardinality-based optimization techniques are not suitable for streaming information sources. In rate-based optimization, fundamental statistics used are estimates of the rates of the streams in the query evaluation tree rather than the sizes of intermediate results. Query optimization only gives the best order in which to evaluate the operators.

References

[1] K. Arnold, A. Wollrath, B. O'Sullivan, R. Scheifler, and J. Waldo. *The Jini specification*. Addison-Wesley, Reading, MA, USA, 1999.

[2] S. Avancha, A. Joshi, and T. Finin. Enhanced Service Discovery in Bluetooth. *IEEE Computer*, 35(6):96–99, June 2002.

[3] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan. Adaptive and dynamic service composition in eflow. Technical Report, HPL-200039, Software Technology Laboratory, Palo Alto, CA, march 2000.

[4] D. Chakraborty, F. Perich, S. Avancha, and A. Joshi. Dreggie: A smart service discovery technique for e-commerce applications. In *Proc. Workshop on Reliable and Secure Applications in Mobile Environments, 20th Symposium on Reliable Distributed Systems*, October 2001.

[5] D. Chakraborty, F. Perich, A. Joshi, T. Finin, and Y. Yesha. A reactive service composition architecture for pervasive

computing environments. In *7th Personal Wireless Communications Conference (PWC 2002)*, Singapore, October 2002.

[6] H. Chen. Developing a Dynamic Distributed Intelligent Agent Framework Based on the Jini Architecture. Master's thesis, University of Maryland Baltimore County, January 2000.

[7] H. Chen, A. Joshi, and T. Finin. Dynamic sercice discovery for mobile computing: Intelligent agents meet jini in the aether. *Baltzer Science Journal on Cluster Computing*, 4(4), Oct 2001.

[8] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An architecture for a secure service discovery service. In *Fifth Annual International Conference on Mobile Computing and Networks (MobiCom '99)*, pages 24–35, Seattle, 1999.

[9] U. D. Discovery and I. platform. World Wide Web, `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.PDF`.

[10] U. B. C. S. Division. `http://ninja.cs.berkeley.edu`.

[11] K. Erol, J. Hendler, and D. Nau. Htn planning: Complexity and expressivity. In *Proc. AAAI.*, 1994.

[12] E. Guttman, C. Perkins, J. Veizades, and M. Day. "rfc2068: Service location protocol, version 2". ftp://ftp.isi.edu/in-notes/rfc2608.txt, 1999.

[13] Hewlett-Packard. *E-Speak Architectural Speciification*, version beta 2.2 edition, December 1999.

[14] E. Houstis, S. Weerawarana, A. Joshi, and J. R. Rice. The PYTHIA project. In S. K. Aityan et al., editor, *Neural, Parallel, and Scientific Computations*, pages 215–218. Dynamic Pub., 1995.

[15] A. Joshi, T. Drashanksy, J. Rice, S. Weerawarana, and E. Houstis. Multiagent simulation of complex heterogeneous models in scientific computing. *IMACS Math. Comp. Simulation*, 44, 1997.

[16] K. Kalpakis, , K. Dasgupta, and P. Namjoshi. Maximum lifetime data gathering and aggregation in sensor networks. In *Proceedings of the 2002 IEEE International Conference on Networking (ICN'02)*, pages 685–696, Atlanta, August 2002.

[17] H. Kargupta and B. Park. Mining decision trees from data streams in a mobile environment. In *Proceedings of the IEEE International Conference on Data Mining (to appear)*, 2001.

[18] Y. Kostov and G. Rao. Low-cost optical instrumentation for biomedical measurements. *Review of Scientific Instruments*, 71(12):4361–4373, December 2000.

[19] Liang Xu. Using Jini and XML to build a component based distributed system. Technical report, University of Maryland Baltimore County, 2000.

[20] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *ICDE*, 2002.

[21] S. Madden, M. J. Franklin, J. M.Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *5th Annual Symposium on Operating Systems Design and Implementation(OSDI)*, Boston, December 2002.

[22] D. Mennie and B. Pagurek. An architecture to support dynamic composition of service components. Systems and Computer Engineering. Carleton University, Canada.

[23] R. S. Patil, R. E. Fikes, P. F. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches. The DARPA knowledge sharing effort: Progress report. In B. Nebeld, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR92)*, pages 777–788. Morgan Kaufman, 1992.

[24] P.Bonnet, J.Gehrke, and P.Seshadri. Towards sensor database systems. In *Conference on Mobile Data Management*, 2001.

[25] S. O. A. Protocol. World Wide Web, `http://msdn.microsoft.com/workshop/xml/general/SOAP_White_Paper.asp`.

[26] Rekesh John. UPnP, Jini and Salutaion - A look at some popular coordination framework for future network devices. Technical report, California Software Labs, 1999. Available online from.

[27] P. Seshadri, M. Livny, and R. Ramakrishnan. Seq: A model for sequence databases. In *ICDE*, pages 232–239, 1995.

[28] S. Viglas and J. F. Naughton. Rate-based query optimization for streaming information sources. In *SIGMOD Conference*, 2002.

[29] D. E. Wilkins and M. desJardins. A call for knowledge-based planning. *AI Magazine*, 22(1):99–115, 2001.

[30] H. C. X. Luan. A meta protocol for agent communication. Technical report, University of Maryland Baltimore County, 1999.

[31] X. Zeng, R. Bagrodia, and M. Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. In *12th Workshop on Parallel and Distributed Simulations*, Canada, May 1998.