DELIS
Dynamically Evolving, Large-scale Information Systems

Integrated Project
Member of the FET Proactive Initiative Complex Systems

Deliverable D2.4.5

Evaluating Data Overlays – Methods and Results
Start date of the project: January 2004
Duration: 48 months
Project Coordinator: Prof. Dr. math. Friedhelm Meyer auf der Heide
Heinz Nixdorf Institute, University of Paderborn, Germany
Due date of deliverable: December 2006
Actual submission date: January 2007
Dissemination level: PU – public

Work Package 2.4: Integrated Application Testbed (IAT)
Participants: RadioLabs (RAL), Universita di Roma “Tor Vergata”, Italy
Authors of deliverable: Giovanni Cortese(g.cortese@computer.org)
Fabrizio Davide (fabrizio.davide@telecomitalia.it)
1 Introduction

WP 2.4 combines an implementation perspective, aimed at the development and evaluation of components related to large scale data management (in this respect being strongly inter-related with WP2.3), while at the same time researching evaluation methods and techniques, which are suitable for understanding the behaviour of large scale, overlay based applications. Evaluation methods should allow to analyze the properties of large scale, self-organising IT infrastructures, using both traditional measurement approaches and non-conventional evaluation approaches based on conceptual tools borrowed from complex systems research.

This report describes two activities we carried out under this framework.

Data Aggregation

We designed an aggregation service, as a building block of a DHT-based data overlay functionality. The aggregation service supports aggregation queries for system-wide performance metrics (such as average or extremal values for load) thus making easier data collection for system evaluation.

Decentralized metrics estimation

We provide in this deliverable a first report of a research in progress into methods for practical collection and computation of global, system-wide performance metrics in distributed networks.

In the sections below we describe the individual research topics which have been investigated. Full technical reports are provided in the DELIS TR series and referenced from this deliverable.

2 Data Aggregation

Data aggregation, in its most basic definition, is the ability to summarize information. Data aggregation is highly relevant to distributed systems, which collect and process information from many sources, like Internet-scale information systems, and peer-to-peer data management systems.

Several architectures and techniques have been recently explored (either in the context of Internet services, peer-to-peer, or wireless sensor networks research), aiming at the design of general purpose, highly scalable data aggregation services. In [DELIS-TR-0463], we first describe application scenarios, and review and compare several proposed designs for aggregation services. Then, we present a complete case study for aggregation services, in the context of a DHT-based peer data management architecture. We argue that aggregation services would benefit for a design which is consistent with and integrated into a general purpose data indexing functionality, and present GREG, an architecture which is based on this assumption.

2.1 Requirements for data aggregation in distributed systems

We first analyze the different application scenarios, which motivate research in data aggregation techniques, and survey the architectural approaches, sometimes quite different, that are documented in the literature.

We identified at least four application scenarios, namely

- Systems Monitoring - Networks of routers, computing nodes (e.g. GRIDS), decentralised Internet applications (e.g. e-mail systems, process management infrastructures etc) are all examples of large, decentralized systems requiring monitoring.

- Self Managing Overlays - Overlay-based architectures are especially relevant to the design of large-scale applications with no clear point of centralized control. The interest in overlay-based architectures is their potential as an enabler for the realization of ‘autonomic’ or self-managing services, running on many computing nodes, which are scalable with respect to the degree of required human intervention, as the number of nodes increases. However, critical to this goal is the ability for each node to obtain a (fairly) complete view of the other nodes’ status.
• 'Querying the Internet' - Another relevant application area is peer-to-peer databases, whether Internet-based as the title of this section evokes, or deployed in large enterprise networks. In a peer-to-peer database, many nodes cooperate to implement storage, indexing and query processing facilities. Peer-to-peer databases should support complex queries, as their centralized counterparts, including range queries and aggregation queries.

• Physical Systems Monitoring (Sensor Networks) - Rather different motivations drive research on data aggregation in sensor networks. Due to size and battery power limitations, these devices typically have limited storage capacity, limited energy resources, and limited network bandwidth. Data produced by nodes in the network propagates through the network via wireless links. When compared to local processing of data, wireless transmission is extremely expensive. The limited amount of energy, bandwidth, and storage capacity available to sensor nodes calls for specialized optimizations of queries injected into the network. For this reason, researchers explored in-network aggregation as a power-efficient mechanism for collecting data in wireless sensor networks.

In [DELIS-TR-0463] we also provide an extensive analysis of the different techniques and architectures that have been developed to face different classes of requirements, outstanding work being represented by such systems such as Astrolabe [RBV03], Tiny Aggregation Service [MFHH02] in the area of sensor networks, the Scalable Distributed Information Management System [YD04], and research on anti-entropy based methods [JM04].

2.2 GREGs: Aggregation-Capable Index Objects

We designed an architecture, based on aggregation and indexing-capable objects named GREGs, which supports aggregation and is consistent with XDM peer-to-peer data management system, described in deliverable D2.4.3. In short, XDM is a DHT based data management system, for storage and query of XML data, which realizes an inverted index scheme on top of a DHT.

GREG is designed both to support ad-hoc queries and long-running queries. However, we assume that techniques to improve aggregate processing are likely to be most important to a scenario, where a) a query runs continuously, over extended periods of time and b) many clients are interested to share the query results. Our architecture is designed for configurations where an important number of nodes are consumers of aggregated information, and consumers are interested to aggregation calculated over filtered views of the data available in the system (e.g. temperature data for office rooms in a site, averaged based on grouping rooms by their floor). In many cases, aggregation must be computed in a hierarchical fashion (e.g. find the average load in increasingly large portions on the network), at different resolutions, and should follow some criteria of proximity (e.g. find the least loaded node among those in my LAN).

In the approach we follow, supporting efficient aggregate calculation requires a preliminary step, whose purpose is close to that of index creation in traditional data management. The setup of an aggregation index involves instantiating several aggregation-capable index objects (GREGs), on different, randomly chosen, peers.

As in traditional indexing schemes, GREGs work by partitioning the data space, according to some application-defined scheme. This partitioning is reflected in two aspects of the index topology:

- bounding predicates of each index node
- links between index nodes GREGs are responsible for calculating one or several aggregation function(s) over the covered partition.

The following is a summary of the characteristics of our design

- GREGs extend a data indexing framework with aggregation capabilities
• GREGs are programmatically created, either under responsibility of some user or of an autono-
mic system optimizer

• A set of GREGs, which collectively realize an index over some attribute, can be arranged in a
graph, typically in a hierarchy

• GREGs can use several strategies for controlling (re)calculation of the aggregate value (data-
driven, query-driven, lazy etc), hence being able to best match the needs of application in terms
of information quality vs. processing cost

• GREGs are themselves DHT-addressable objects

2.2.1 Query Model and Aggregates in Queries

Here we describe the characteristics of the aggregate abstraction as seen by the clients of the system.
Logically, an aggregation query consists of the following.

• An aggregation clause consisting of a function over one or more aggregation attributes or some
expression calculated on attributes. Arbitrary aggregation functions can be specified

• A selection, defining the set of data which should actually be input to aggregate calculation
(e.g. compute aggregation function over all performance events from agents e.g. where role is
'backbone router')

• An aggregation dimension, i.e. the attribute(s) used for grouping data into sets, over which
intermediate aggregate is calculated, and the associated partitioning rules. In the simplest case,
a 1-level partition of data values can be used, as in TAG (e.g. compute the aggregate over all
performance events from servers, grouped by cpu-type). More complex, range-based or datacube
style roll-up capabilities can be obtained by defining partitioning rules that yield respectively
a range-based or a hierarchical decomposition of the data in the aggregating dimension.

• The cardinality in the time dimension of the aggregate. While in some cases the aggregate is
a single value wrt time (i.e. it represent the current MAX/ MIN/ AVG etc), in other cases its
value must be a time series of values

• An update strategy governing whether the update of the aggregate value is data driven or
request driven Evaluation of an aggregate query yields multiple values; individual values repre-
sent the value of the aggregate for different partitions, or ranges, of the data space, according
to the aggregation dimension, and the value of the aggregate for the partition at different time
instants i.e. for different epochs. Aggregate computation is performed in a distributed style
over several nodes (each associated to a partition of the data), as described below. GREG
functionalities We start by covering the behavior of GREGs with respect to indexing. As intro-
duced, each GREG is responsible for (or covers) some value range, of the indexed attributes.
The interval is defined by a bounding predicate associated to the GREG. GREGs have two
fundamental behaviors. First, they support indexing and retrieval of persistent data by storing
pointers to data objects. Second, they support data dissemination by acting as a rendez-vous
point for data matching the their bounding predicate. Leaf GREGs store a reference to each
indexed object. In addition, they implement a cache which may store a set of attributes of
the indexed object, for performance reasons. In addition to leaf index nodes, additional nodes
(interior or root nodes) can be created that cover increasingly larger portions of the attribute
value range. With respect to aggregation, GREGs provide several functionalities

• they maintain partial state needed to compute the aggregate;
for epoch-based, continuous aggregation, they manage the required synchronization logic among children, before computing the aggregate;

- they support propagation logic as required in the aggregation query e.g. manage (upward) propagation of updates in the GREG tree as well as pushing downward requests of aggregate recalculation.

### 2.2.2 GREG functionality

GREGs have two fundamental behaviors. First, they support indexing and retrieval of persistent data by storing pointers to data objects. Second, they support data dissemination by acting as a rendezvous point for data matching the their bounding predicate.

With respect to indexing each GREG is responsible for (or covers) some value range, of the indexed attributes. The interval is defined by a bounding predicate associated to the GREG. Leaf GREGs store a reference to each indexed object. In addition, they implement a cache which may store a set of attributes of the indexed object, for performance reasons. In addition to leaf index nodes, additional nodes (interior or root nodes) can be created that cover increasingly larger portions of the attribute value range. Intermediate nodes provide aggregation functionality

- they maintain partial state needed to compute the aggregate;

- for epoch-based, continuous aggregation, they manage the required synchronization logic among children, before computing the aggregate;

- they support propagation logic as required in the aggregation query e.g. manage (upward) propagation of updates in the GREG tree as well as pushing downward requests of aggregate recalculation.

Finally, GREG support dissemination of (aggregate) data. Dissemination of aggregate values to interested clients is a separate process from aggregation. The process is similar to that used for normal XDM queries. A query parser process a subscription query and recognizes the index object(s) which are candidate to support it. The client joins a multicast tree rooted at the index object, which acts as a rendez-vous point for all the data matching the bounding predicate of the index object. This is currently implemented by leveraging the Plaxton-tree based multicast features built into Pastry/Scribe. Other techniques for dissemination might be used as well.

### 3 Decentralized Metric Estimation

Decentralized environments, as Peer to Peer (P2P) networks, are increasingly spreading through the Internet. Their open structure indeed offers the possibility of sharing a huge quantity of information and resources. The main problem of these kind of networks is the lack of a central authority that can provide information about the performance of individual peers, or guarantee and certify the quality of the shared resources. The concept of shared resource or performance is very general here. In fact, it can encompass both application-level behavior of a peer node (or even of its user), such as quality of information the peer is storing, or infrastructure-level behavior, such as the bandwidth the peer has available, the response time in performing a certain transaction and the like. We seek mechanisms that allow peers of a network to collect information and estimate metrics, describing the performance and quality of other peers, without resorting to a centralized service. To fit with the characteristics of peer-to-peer networks, where individuals or coalition of peers may alter the measurements to obtain some advantage, such mechanisms should be robust against malicious peers. Our aim is to enable a distributed algorithm that is run by autonomous untrusted agents to be relatively reliable, efficient and secure.
In the kind of applications we are interested in, we are often concerned with assisting a peer, which needs to engage with another peer to perform some kind of transaction, in the selection of a peer which has the desired characteristics with respect to some metric. Also, we need to assist him with the rating of the information coming from other peers. The approach we take shares clearly a lot with other research on reputation management.

The approach we take combines ideas from EigenTrust/TrustRank. As in EigenTrust, peers rate each other after each transaction (e.g., download of a file in a file sharing network, invocation of a service in a distributed computing system), thus building a local trust value of any other peer they have transacted with. [KSGM03] aggregates the local trust values of all of the users using the notion of transitive trust: A peer i will have a high opinion of those peers who have provided it authentic files. Moreover, peer i is likely to trust the opinions of those peers, since peers who are honest about the files they provide are also likely to be honest in reporting their local trust values. The idea of transitive trust leads to a system where global trust values correspond to the left principal eigenvector of a matrix of normalized local trust values. The EigentTrust work also suggests several practical ways to computing normalized local trust values and global trust values.

Our method uses a slightly modified version of EigenTrust, Inverse GlobalTrust, applied to the inverse transition matrix and, starting from this, two combined approaches based on two algorithm introduced in [BCD+06] for detecting Web spam. Our mechanisms result effective for reducing the number of wrong selections of a peer (e.g., inauthentic downloads) in two cases in which EigenTrust alone is not sufficient. The first case, known as threat model D, was originally introduced in the [KSGM03]. Here the malicious peers are of two different types: Part of them always uploads inauthentic files and assigns positive values only to malicious peers, the others, called malicious spies, answer 0.05% of the most popular queries always uploading authentic files but assign trust value only to malicious peers. In this way, the spies gain trust local value that are able to somehow transfer to malicious peers.

We have built a simulator of our model, customized for a file sharing application. Initial results of the simulation show that our modified version of EigenTrust is able to reduce the number of inauthentic download from 35% (using only EigenTrust) to 3%. We present a more sophisticated attach, that we call, following [KSGM03], Threat Model G, in which malicious peers to further bedim their presence in the network, start to assign a local trust value also to a small fraction of good peers. Inverse GlobalTrust performs well also in this second case, reducing the downloads of inauthentic files to ~8 ÷ 5%. The combined approach behaves even best: Upon activation of our scheme, around the 0.5% of all file downloads return an inauthentic file.

The model is general enough to be applied to several other types of services, for estimating metrics characterizing service performance of a peer in a distributed application. Our plan is to generalize the simulator as well as the method to other types of applications, including peer-to-peer data management systems of the kind of our XDM platform. Our report describes metrics to be estimated in the context of a p2p data management system.

4 Main contributions

The main contribution to section 2 is [DELIS-TR-0463].

Details for research summarized in section 3 can be found in [DELIS-TR-0470].

References

[DELIS-TR-0463] Fabrizio Davide Antonino Virgillito Roberto Beraldi Vivien Quema Giovanni Cortese, Federico Morabito. Data Aggregation in Large Scale Distributed


