Towards a Grid-Based Zero-Latency Data Warehousing Implementation for Continuous Data Streams Processing

Tho Manh Nguyen, Vienna University of Technology, Austria
Peter Brezany, University of Vienna, Austria
A Min Tjoa, Vienna University of Technology, Austria
Edgar Weippl, Vienna University of Technology, Austria

ABSTRACT

Continuous data streams are information sources in which data arrives in high-volume, in un-predictable rapid bursts. Processing data streams is a challenging task due to (1) the problem of random access to fast and large data streams using present storage technologies, (2) the exact answers from data streams are often too expensive. A framework of building a Grid-based Zero-Latency Data Stream Warehouse (GZLDSWH) to overcome the resource limitation issues in data stream processing without using approximation approaches is specified. The GZLDSWH is built upon a set of Open Grid Service Infrastructure (OGSI)-based services and Globus Toolkit 3 (GT3) with the capability of capturing and storing continuous data streams, performing analytical processing, and reacting autonomously in near real time to some kinds of events based on well-established Knowledge Base. The requirements of a GZLDSWH, its Grid-based conceptual architecture, and the operations of its service are described in this paper. Furthermore, several challenges and issues in building a GZLDSWH such as the Dynamic Collaboration Model between the grid services, the Analytical Model, the Design and Evaluation aspects of the Knowledge Base Rules are discussed and investigated.

Keywords: Grid based Zero-Latency DWH, Data Streams processing, dynamic Workflow execution, OLAP Cube Management.

INTRODUCTION

We are entering a new area of computing in today’s complex world of computational power, very high speed machine processing capabilities, complex data storage methods, next generation telecommunications, new generation operating systems and services, and extremely advanced network services capabilities. At the same time, the number of emerging applications which handle various continuous data streams (Babcock B. et al.,2002; Chandrasekaran S. & Franklin M., 2002; Widom J. et al, 2003; Stonebraker M. et al. 2003; Lerner A. & Shasha D., 2003), such as sensor networks, networking flow analysis, telecommunication fraud detection, e-business and stock market online analysis, is growing.

It is demanding to conduct advanced analysis over fast and huge data streams to capture the trends, patterns, and exceptions. However, to fully extract the latent knowledge inherent within the huge data is still challenging effort because of the existing insufficient technology. Data streams arrive in high-volume, in un-predictable rapid bursts, and need to be processed continuously.

Processing data streams, due to the lack of resources, is challenging in the following two respects. On the one hand, random access to fast and large data streams is still impossible in the near future. On the other hand, the exact answers from data streams are often too expensive. Therefore, the approximate query results (Babcock B. et al.,2002; Guha S. & Koudas N., 2002; Dobra A. et al., 2002; Tucker, P., Maier D., & Sheard T.,2003; Widom J. et al, 2003; Kim J. & Park S.,2005) are still acceptable because there is no existing computing capacity powerful enough to produce exact analytical result on continuous data streams.

The significant increased data volume of information manipulated in several domains has affected Data Warehousing (DWH) and Business Intelligence (BI) applications. Data Warehousing and Business Intelligence applications are normally used for strategic planning and decision making.
However, existing DWH technologies and tools (e.g. ETL, OLAP) often rely on the assumption that data in the DWH can lag for a tolerable time span (e.g. on a few hours) behind the actual operational data and the decisions are based upon the analytical process on that “window on the past”. For many business situations, especially, in data stream analysis, this decision making approach is too slow due to the fast pace of today’s business. Today’s decisions in the real world thus need more real-time characteristics and consequently DWH, BI, ETL tools and OLAP systems are quickly beginning to incorporate real-time data. A new ETL approach using widely accepted Web technologies has recently been announced (Schlesinger L. et al., 2005).

Starting from the concept of a Zero-Latency Data Warehouse (ZLDWH) (Bruckner R., 2002; Tho N. & Tjoa A., 2003), we extend the system to tackle continuous data streams with the capability of capturing data streams, performing analytical processes, and reacting automatically to some kinds of events based on well-established knowledge.

We do not follow the approximation approach; instead, we capture and store all data streams continuously while performing the analytical processing. Obviously, such systems require a very high computing capacity which is capable of huge storage and computing resources. Fortunately, in the last few years we have witnessed the emergence of Grid Computing (Foster I., Kesselman C., & Tuecke S., 2001; Joseph J. & Fellenstein C., 2003) as an important new technology accepted by a remarkable number of scientific and engineering fields and by many commercial and industrial enterprises. Grid Computing provides highly scalable, secure, and extremely high performance mechanisms for discovering and negotiating access to remote computing resources in a seamless manner. With the Grid as “a flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources”, our requirements seem to be satisfied. Furthermore, due to unpredictable characteristics of data streams, Grid technology is more convincing due to its flexibility.

This paper describes our ongoing work in developing the GZLDSWH built upon a set of OGSA-based grid services and GT3 toolkit. The GZLDSWH is composed of several specific Grid services for capturing, storing, performing analysis on continuous data streams and issuing relevant actions or notifications.

This paper is organized as follow. First, we mention other research work related to the paper. Next, the GZLDSWH system overview will be described. We then discuss several approaches in building the GZLDSWH and specify its conceptual architecture. The next section describes the operation of GZLDSWH in conducting analysis processes and reaction on continuous stream. Thereafter, we focus on the main contributions of our paper:

- A model that describes the required dynamic collaboration of the grid services.
- A Grid-based OLAP Cube management service.
- The Knowledge Base rules and rule evaluation process.
- A guideline on how to increase performance when nodes fail and need to roll back to previous checkpoints.

Our ongoing prototype implementation will be described briefly afterwards. Finally, we give our conclusion and mention the future work.

RELATED WORK

ZLDWH requires several extended features compared to the traditional DWH (Bruckner R., 2002). Firstly, the traditional batch snapshot approach to extract source data must be replaced by processes that continuously monitor source systems and capture data modifications as they occur, and then load those changes into a DWH. Secondly, not only the continuous data integration, but also the real time automatic analysis engine is necessary to make the DWH more active.

There are several approaches from both academia and the industrial community for realizing zero-latency information systems. Compaq’s Zero Latency Enterprise (ZLE) framework (Compaq Corp., 2002) is centered around an operational data store (ODS) as the primary data repository instead of a DWH. NCR’s Teradata division has developed the concept of the @ctive Data Warehouse (Teradata Corp., 2002) which essentially marries the operational data store (ODS) and DWH concepts.

Active rules (Bertino, E. et al., 2000; Cho E. et al. 2003, Bailey, J.; Poulavassilis, A.; Wood, P., 2003) have been widely accepted to achieve the goal of
auto-decision-making. Research in active databases (ADBs) (Paton W. & Diaz O., 1999) extends the power of active rules to react to events and conditions in the database. Active Data Warehouses (ADWHs) (Thalhammer T. & Schrefl M. 2002) are systems which use event-condition-action rules (ECA) or other event-driven mechanisms in order to carry out routine decision tasks automatically within a DWH environment.

Thalhammer adopted ECA rules to mimic the work of an analyst, so he calls them analysis rules. His approach combines ADBs, DWHs, and OLAP to automate decision processes for which well-established decision criteria exist. However, the data integration process is based on traditional batch loading and concerns that the Data Warehouse state remains constant during a cycle.

Qtool (Bruckner R., 2002), Bruckner’s solution for zero-latency data warehousing, is based on the continuous data integration using Microsoft Message Queuing (MSMQ) and Teradata Tpump utilities. Although data are loaded and integrated continuously, QTool does not deal with other problems such as feeding data from heterogeneous data sources, detecting data changes, active rule modeling or implementing the active decision engine. Obviously, these tools are not designed to support continuous data streams processing.

Data streams have some specific characteristics that make them different from traditional data. They could be “infinite”, and once a data element has arrived, it should be processed and either archived or deleted, i.e. only a short history can be stored. It is also preferable to process data elements in the order they arrive, because sorting even sub streams of a limited size, is also a blocking operation. So far, research results have been reported for modeling and handling data streams including algorithms for data stream processing to full-fledged data stream systems.

In continuous query processing, several approximation methods are used for data reduction and synopsis construction such as sketches (Dobra A. et al., 2002), random sampling, histograms (Muthukrishnan S. & Strauss M. (2003), and wavelets (Chakrabarti K. et al., 2001). Some other approximation methods are applied to tackle the blocking operator such as Sliding Window (Chandrasekaran S., & Franklin M., 2002), load shedding (Babcock B., Datar M., & Motwani R., 2004), punctuation (Tucker, P., Maier D., & Sheard T., 2003). “k-Constraints” (Babcock B. & Olston C., 2003) are used in clustering and monitoring data streams. Kim J. & Park S. (2005) propose an efficient periodic summarization method with a flexible storage allocation to store large volumes of streaming data in stable storage. Other research topics cover data stream management system models, architectures and related issues such as memory minimization, operator scheduling, query optimization, multiple query, distributed query processing and so on (Babcock B. et al. 2002, Widom J. et al, 2003).

As another approach, conventional OLAP and data mining models have been extended to deal with data streams, such as multi-dimensional analysis (Han J. et al., 2002), clustering (Motvani R. et al., 2000) and classification (Hulten G., Spencer L., & Domingos P., 2001). However, most of previous approaches on data stream processing focus on approximation methods based on statistical estimations due to the limited storage and computing resources. Our effort, instead, tries to store all data streams and process them within the Grids as if they are stored in one extremely large distributed database.

In recent years, Grid computing (Foster I., Kesselman C., & Tuecke S., 2001; Joseph J. & Fellenstein C., 2003; Foster I., & Grossman R., 2003) is emerging as the best solution to the problems posed by the massive computational and data handling requirements. Starting from the concept of linking super computers to benefit from the massive parallelism for computation needs, Grid’s focus has recently shifted to more data-intensive applications where significant processing is conducted on very large amounts of data.

New-generation Grid technologies are evolving towards an Open Grid Services Architecture (OGSA) (OGSA, 2003) in which a Grid provides an extensible set of services that virtual organizations can aggregate in various ways. The development of OGSA technical specification is in progress within the Global Grid Forum covered by the tasks called the Open Grid Services Infrastructure (OGSI) with Globus Toolkit 3 (GT3) (Sotomayor B., 2004). The Database Access and Integration Services (DAIS) Group developed a specification for a collection of OGSI-Compliant Grid database services. OGSA-DAI Release 3 (GGSA-DAI, 2003), the first
The reference implementation of the service interface is already available. So far, only little attention was devoted to knowledge discovery on the Grid. An attempt to design the architecture for performing data mining on the Grid was presented in (Cannataro M. & Talia D., 2003). The authors present the design of a Knowledge Grid architecture based on the non-OGSA-based version of the Globus Toolkit, and do not consider any concrete application domain. R. Moore presents the concepts of Knowledge-Based Grids in (Moore R., 2001) and Semantic Grid explored by (Roure D., Jennings N., & Shadbolt N., 2003) towards a knowledge-centric and metadata-driven computing paradigm. The WP4 of the OGSA-DAI project addresses the design of a distributed query processing service for the Grid. Recently, GridMiner (Brezany et al., 2003; Kickinger et al. 2003) has been reported as an evolution of parallel and distributed data mining technology and Grid Database Services development.

Workflow, “the coordinated execution of multiple tasks or activities” (Marinescu D., 2002), can be extended and applied virtually to other areas, from science and engineering to entertainment. Web services already provide mechanisms to handle complex workflows. Since every Grid service is a Web service with improved characteristics and services (Sotomayor B., 2004) (the converse of this statement is not true), it is possible to adapt the ideas for workflow compositions from Web services and apply them to Grid services.

BPEL4WS 1.1 (IBM, 2003) is the actual standard, which describes compositions of Web Services. The Grid Services Flow Language (Krishnan, S., Wagstrom P., & Laszewski G. 2004) intends to do the same for Grid Services. GSFL is based on the so called Web Services Flow Language (Leymann F., 2003), a predecessor of BPEL4WS, published by IBM. All of these flow language specifications have the same target: describing a business process built up of various web services. This description then serves as input for a workflow engine like BPWS4J (IBM, 2003) (an engine for BPEL4WS developed by IBM). Such an engine works with the persistent Web services, and it requires the specification documents of “physical” Web service URLs. However, in the ZLGDSWH system, the services are transient Grid services which will be created on demand. Several workflow solutions such as Triana (Taylor I., 2003), Pegasus (Kesselman C. et al., 2004), GridFlow (Cao J. et al., 2003), GridPhyN (Deelman E. et al., 2003), McrunJob (Graham G., Evans D. & Bertram I., 2003) are used to create and manage the grid computational workflow. However, the above projects did not take into account the automatic collaboration between the Grids services. Because of its “automated event-based reaction” feature, GZLDSWH requires the higher level of automation in service creation, discovery, invocation and destruction according to the Grid environments.

THE GZLDSWH OVERVIEW

Figure 1. The overall process of the Grid-enabled Zero-Latency Data Warehouse System

Starting with the idea of building a Zero-Latency analytical environment dealing with heterogeneous data sources, we extend the system to conduct analysis on continuous data streams. A ZLDWH (Bruckner R., 2002; Tho N. & Tjoa A., 2003) aims to significantly decrease the time to react to business events allowing the organizations to deliver relevant information as fast as possible to applications which need a near real-time action to new information captured by an organization’s information system. It enables analysis across corporate data sources while still continuously update the new arriving data and notifies the handling of actionable recommendations, alerts or notifications. In Data Stream applications, events take the form of continuous data streams.

The exact analysis results on these data stream events are very expensive because they require high computing capacity which is capable of huge storage and computing resources. Therefore, a Grid-based
approach is applied in ZLDWH to tackle the lack of resources for continuous data stream processing. Figure 1 depicts significant phases throughout the overall process of such GZLDSWH. Continuous data streams will be captured, cleaned and stored within the Grids. Whenever the analytical processes need to be executed, immediately after the arrival of new data or based on predefined timely scheduling, the virtual Data Warehouse will be built on the fly (Foster I., & Grossman R., 2003) from data sources stored in the Grid nodes. Obviously, this approach is not concerned with traditional incremental updating issues in Data Warehouse because the virtual DWH is built from scratch using the most current data. Analytical processes will then be executed on such virtual DWH and the results will be evaluated with the use of the Knowledge Base. Finally, dependent on the specification of the Knowledge Base, the system sends notifications, alerts or recommendations to the users.

There are a significant number of applications in which the conducting of analytical processing on continuous data streams is necessary for detecting trends and abnormal activities. The following scenario describes an example in Mobile Phone Fraud Detection although the usage of such system could also be applied in other time critical decision support applications.

Expert users defined rule specifications for fraud detection, e.g. “if an international mobile call lasts over 1 hour it will be considered as a fraudulent call”. The rule could also be more complex such as “an international mobile call from Austria to Vietnam of a certain customer lasts over 30 minutes will not be considered as a fraud, if its duration is not over 1.5 times of his/her average call duration from Europe to Asia within the last 3 months, otherwise, it is considered as a fraudulent call”. These rules are stored in the Knowledge Base and are referenced by the Rule Evaluation module before it makes the final decisions. Expert users could also specify how and when the system operates to detect the Fraud situation by submitting the pre-defined plan workflow. In this plan, the experts specify the order of module executions and the time point when they should be executed. The whole system operation will then be monitored and controlled by the Workflow Control module which follows the pre-defined plan.

When the end-user makes a phone call, the Call Detail Records (CDRs) are issued continuously as continuous data streams. Because of the special characteristics of continuous data streams, these CDRs must be captured and stored in timely fashion. The data storage is heterogeneous and geographically distributed in several grid nodes.

For supporting analytical processing, the virtual DWH is built on the fly from the heterogeneous, distributed data sources as follows. The OLAP server accesses the raw data items from multiple Grid nodes and creates the pre-aggregated data cubes. The Data Mediator allows the OLAP server to access the distributed, heterogeneous data sources as if they were local data sources. In some situations, Data Preprocessing could be necessary to clean the data, standardize the data, or transform data into the common format before storing into the OLAP cubes.

When the virtual DWH data is available, the OLAP server accepts the queries from Data Analysis or Data Mining tools, executes these queries and returns the results. To detect a fraud situation, it is necessary to analyze the CDRs at multiple levels with different dimensions and in a variety of time ranges e.g. to calculate the average international mobile call length from Europe to Asia within the last 3 months of a certain customer. The OLAP server therefore accepts analytical commands such as “drill up”, “drill down”, “slide and dice” and performs these operations on the data cubes.

The results are returned to the Rule Evaluation module, and it will access the Knowledge Base to evaluate the rules. If some pre-defined criteria in the Knowledge Base are satisfied, the Rule Evaluation module performs suitable actions, for instance, sending of notifications to the users, or stopping the telephone services. Particularly, if the rule criteria
lead to an ambiguous situation, the Rule Evaluation module would issue other analytical queries to further investigate the data.

The above scenario highlights following requirements for a GZLDSWH system:

- **Knowledge-Base Rule Support**: The final decisions, e.g. whether a phone call is fraudulent or not, is based on the set of Knowledge-Base rules specified by expert users. The Knowledge-Base Rules preserve the experiences, and knowledge to drive the decision support process. It is necessary to develop a component that allows the expert users to easily manipulate the existing rules in the Knowledge Base i.e. insert, delete or update rules. The autonomous validation and consistency checking of these rules should also be considered. The facilities of auto-inference and auto-learning are also the challenges of Knowledge-rule management.

- **Multi-level Analysis Support**: In order to evaluate the rules, it is necessary to conduct analytical process on the multi-dimensional historical transaction of customer CDRs to identify the customer’s pattern. Besides, in some situations, there is the uncertainty or ambiguity in evaluating the rules during online analysis process. Then it is necessary to conduct analytical process on the whole CDRs at multiple levels to define the final decision. CDR streams thus need to be stored without loss within the Grid. DWH repositories and online analytical processing (OLAP) cubes are built on the fly from these Grid node’s data. The significant parts are: (1) the creation and maintenance the OLAP Cube, (2) the OLAP query engine that executes analytical queries on OLAP data, and optionally (3) the OLAP Data Mining Engine for execution of the on-line analytical mining (OLAM) algorithms.

- **Automated Reaction Support**: Whenever the Fraud is detected via CDRs analysis and rule-based evaluation, the system must have the ability to issue automatically the relevant actions respective to the Fraud prevention methods. That could be the alarm or recommendation message sent to the mobile phone customers, or it could
also be the service disconnection to prevent more fraudulent activities. However, this automatic reaction must follow the rules stored in predefined Knowledge Base.

- **Distributed, Heterogeneous Data Acquisition:** The ability to use different kinds of data sources and data-warehouse repositories is mandatory for Data Grid systems. Therefore, the Data Mediator layer is usually used to transparently access the heterogeneous Grid-based data sources.

- **Transparent User Interface:** The system should be able to provide the users an interface which is easy and transparent to the Grid, Network, and Location. Customers can access the system to register the services or send their feedback from anywhere using a variety of devices (mobile, PDA, laptop, etc.) without considering the complexity of system architecture. Expert users use the designed interface to specify the rules without considering the Grid structure, network feature and the physical details or location of data sources.

- **Flexibility and Open Architecture:** The essence of Grid is heterogeneity. The GZLDSWH thus should allow the integration of components that are heterogeneous i.e. written in different programming languages or are optimized for different platforms. Technical and contextual changes to the underlying data sources, interface implementations, libraries, etc. must not affect the module operability. The components and features shall be easily extensible allowing for plug-ins to be executed. Furthermore, the architecture must be open to integrate specialized third-party toolkits.

- **Grid Environment Information and Monitoring:** The Grid environment always varies significantly during the runtime. Therefore the system should be able to invoke and execute its component dynamically dependent on the runtime environment. For this purpose, the Grid environment information should be monitored by a special component. It maintains the knowledge about the resources available on the Grid, their capacity and current utilization. It should be easily query-able, highly scalable and have to react quickly even under heavy load.

- **Dynamic Component Invocation and Execution:** As described in the scenario, the component execution flows are pre-defined by expert users. However, the Grid environment always varies significantly during the runtime. Therefore the system should be able to invoke and execute its components dynamically depending on the runtime environment.

- **Integrity and Availability:** Assuming that all grid nodes are trusted entities, there still is a need to implement integrity checks and address non-availability. If nodes cannot be trusted, data integrity checks are required. Kamvar et al. (Kamvar D. et al., 2003) provide an excellent overview of requirements and also shows how these issues are addressed in P2P networks. Their solutions can also be adapted to Grid computing if one is prepared to accept the computational overhead.

There are a lot other requirements such as security, high availability, scalability and performance but they will be planned for the future work.

**FRAMEWORK FOR BUILDING A GZLDSWH**

Based on the requirements discussed above there are three approaches for building the GZLDSWH system.

**Building the whole system from scratch**

Following this way, we have to develop not only the components of the system, but also have to deal with many other issues in Grid Computing such as protocol for communication, message passing mechanism, resource management, scheduling, life cycle management, etc. The only advantage is that we do not have to obey any specifications and freely apply any container technology such as J2EE Container, .Net Container to control and manage our components. However, a lot of work must be done, and the system then can not be integrated easily with the Grid Community standard toolkits and specifications. We thus do not choose this approach to develop our system.

**Open Grid Service Architecture (OGSA)**

The second option is to build the system using the Open Grid Service Architecture (OGSA) (OGSA,
2003) based on the Open Grid Service Infrastructure specification (OGSI) and Globus Toolkit version 3 (GT3) (Globus Toolkit, 2003). During the last years the Service Oriented Architecture (SOA) gained popularity as a new software engineering paradigm. It arose from the necessity of creating components providing clearly defined small pieces of functionality that later on can be assembled into complex (usually distributed) applications. The Web Services Model follows the SOA and allows applications to communicate using agreed, widespread standards and protocols independent of their implementation and platform. The Open Grid Service Architecture (OGSA) represents the convergence of Web service and Grid computing technologies with the aim of describing the next generation of Grid Architecture in which the components are exchangeable on different layers. Consequently, in OGSA, all kinds of storage and computational resources, components, databases, file systems, etc are exposed as services. This approach has the big advantage that the upper-layer components have to be concerned only on a small amount of interfaces because the implementation is hidden behind the interface.

As mentioned previously, the Open Grid Service Infrastructure (OGSI) defines specifications with many interesting features such as factory service discovery, instance creation, invocation, lifetime management, notification and manageability. The Globus Toolkit 3 (GT3) implements most of these OGSI specifications and provides us the infrastructure components for resource management, monitoring and discovery, security, information services, data and file management, communication and fault detection. Some features still missing are a resource broker, load balancing and scheduling functionality. However, if we develop the system on top of the OGSI and GT3 toolkit, some of our above mentioned requirements are satisfied, such as Flexibility, Open Architecture, Grid-Network-Location Transparency and Grid Environment Information. Hence we chose this approach to develop the GZLDSWH system.

**WS-Resource Framework**

Around the same time when the OGSI work has been progressing, the Web services architecture has evolved as well, for example, the definition of WSDL 2.0 and the release of new draft specifications such as WS-Addressing (Globus Alliance, IBM, & HP, 2004). WS-Resource Framework can be viewed as a straightforward refactoring of the concepts and interfaces developed in the OGSI V1.0 specification in a manner that exploits recent developments in Web services architecture. The difference is that WSRF uses different constructs for modelling the stateful resources and the stateless Web services, while OGSI uses the same construct (the service instance). The WS-Resource Framework is still new, and the Globus Toolkit version 4 which integrates this framework, is expected to release at the end of 2004. Therefore, in the future, the GZLDSWH system developed upon OGSI/GT3 should be based on the WS-Resource Framework.

**The Grid-based Conceptual Architecture of GZLDSWH**

Each phase of the process in Figure 1 includes several tasks such as capturing, storing data stream, building OLAP cube, conducting multidimensional analysis etc. Sharing the same approach with GridMiner project (Tjoa A. & Brezany P. et al., 2003), each particular task will be realized as a grid service. The GZLDSWH is thus composed of several specific Grid services for capturing, storing, performing analysis on continuous data streams and issuing relevant actions or notifications reflecting the trends or patterns of the data streams. As we have discussed, these grid services are built on top of OGSI and GT3 toolkit and can be grouped into several layers based on their functionality as described in Figure 3.
The Fabric and Core Layer: The services in this layer are provided by the Globus Toolkit 3 which enables most of the basic operations and communication between the Grid Services. We do not need to develop these fabric services and Grid core services (the white box in Figure 3).

The Facilities Layer: This layer provides some services for monitoring the Grid environment, scheduling, load-balancing and so on. It also provides the transparency of resources, network and location to the heterogeneous data sources. The services in this layer include the following:

- **System Information Service (SIS).** The SIS is a vital service within every grid infrastructure, providing static and dynamic information on available grid resources. It is a specialized implementation enabling specific decision making and monitoring processes.

- **Resource Broker Service (RBS).** The Resource Broker is used to find best-fitting resources for resource allocation, such as match-making of the requests and resources. In our system, the Resource Broker is used as a reference for the Workflow Engine in dynamic discovery, creation, binding and invocation of other available services instances in the runtime.

- **Data Mediation Service (DMS).** The DMS provides a single virtual data source having the same client interfaces as classical grid data sources but it integrates data from multiple heterogeneous federated sources from several Grid nodes. This service simplifies access requests and implements transparency for heterogeneous data sources.

The Data Input Layer: This layer provides the services for capturing, cleaning, and storing data within the Grid. The following services belong to this layer:

- **Data Capturing Service (DCS).** The Data Capturing Service works closely with the stream sources such as sensors, cameras, satellites etc., for capturing data streams in the limited time without data loss.

- **Data Cleaning Service (DES).** In some special situation, the Stream Data should be cleaned prior to being stored in the Grid. Therefore, the Data Cleaning Service is an optional service.

- **Data Storing Service (DSS).** The Data Storing Service resides in each Grid node and is responsible for storing the lossless data streams. This service could also convert the data into an appropriate format for the local data source.

- **Data Integration Service (DIS).** This service is responsible for secure, reliable, efficient management and operation of the necessary data transfers within the grid environments.

The Data Analysis Layer: The aim of the Data Analysis layer is to support the analytical process on data stream stored somewhere within the Grid. This layer contains the following services.
• **Data Preprocessing Service (DPS).** The Data Preprocessing Service performs several preprocessing activities such as data cleaning, normalization, selection, reduction, transformation, etc. before storing the data into the OLAP cube.

• **OLAP Cube Management Service (CMS).** The OLAP Cube Management Service is one of the major components of the system. This service creates distributed OLAP cubes from several data sources stored at specified Grids nodes. After the initial cube creation, the service can be used for cube interaction and life cycle management.

• **Data Analysis Service (DAS).** The Data Analysis Service is another major component of the system. It works very closely with the OLAP Cube Management Service and performs analytical process by sending queries and commands such as “drill up”, “drill down”, “slide and dice”. It thus allows analyses of datasets at different levels of abstraction. The output of the Data Analysis Service is the analysis result which will then be evaluated by the Knowledge Base for further actions.

• **Data Mining Service (MIS).** The Data Mining Service is created as an extensible framework providing necessary data mining algorithms making it convenient for the related application developers to easily plug in their algorithms and tools.

• **Rule Evaluation Service (RES).** The RES service takes the analytical results from the DAS, evaluates these results against the Knowledge Base rules, and finally takes suitable actions. It could invoke the NAS to perform suitable actions if the rule criteria are satisfied. It could also invoke the DAS service or the Data Mining service (MIS) to perform further analytical process when the rule criteria are too ambiguous to make the final decision.

• **Notification/Action Service (NAS).** The NAS service manages all possible actions of the GZLDSWH system such as issuing notifications, alarms, and recommendations to the users to inform about the abnormal trends. In addition, it invokes the Analysis service when receiving the local update message.

**The Workflow Control Layer:** contains the Workflow Management Service that controls the dynamic service invocations, executions, and destructions.

• **Workflow Management Service (WMS).** This service allows expert users to specify the logical workflow of system activities and execute complex and highly dynamic workflows for several heterogeneous grid services. The WMS service dynamically controls service execution, service termination and communication, etc. dependent on the Grid environment. Because the Grid environment significantly varies over time, the Dynamic Workflow Control Service is an extremely important component in the system.

**THE OPERATION OF GZLDSWH**

Within the Grids environment as described in Figure 4, there are one Master node and several child nodes (Node 1, 2…, Node N). The Master node controls other child nodes to fulfil system activities. The role of these child nodes is to store data within the Grid environment. The Master node therefore includes most of the essential services while the child nodes only contain some data input services and local data update detection services. The Master node also keeps the Grid metadata for Grids management and the Knowledge Base rules for controlling event reaction behaviour.
The operation plan of the services in GZLDSWH is specified in the logical workflow as described in Figure 4 in which the services are arranged in the following logical order. The Data Capturing Service (DCS) receives continuous data streams from stream sources such as sensor systems, satellites, etc. Due to the huge amount of incoming data, the DCS must capture the data timely and invoke available Data Storing Services (DSS) residing at several child nodes for storing data. The DCS could also invoke Data Cleaning Service (DES) to clean the data before storing if necessary.

After storing data at child nodes, the Analysis Service (DAS) at the Master node will be invoked immediately or after predefined timely schedule depending on application requirements and performance trade off. DAS execution will create the virtual Data Warehouse from scratch. For this purpose initially, the DASs available at several local child nodes are invoked. Due to this, the Cube Management Service (CMS) gains the essential raw data at the child nodes to build the global OLAP cube. Each child node contains part of the cube namely “cube chunk”. Data will then be integrated into the common format by the Data Integration Service (DIS). Before being stored into the virtual Data Warehouse, data can be passed to pre-processing phase via the Data Pre-processing Service (DPS). The DPS can perform several tasks such as data cleaning, data transformation, data normalization or data reduction. After the global cube is formed, the DAS will perform analysis queries or data mining algorithms (via the equivalent Mining Service - MIS) based on the data inside the virtual DWH. The analysis results then will be sent to the Rule Evaluation Service (RES).

The RES accesses the Knowledge Base and evaluates the rules. The Knowledge Base rules are provided by the user through the Rule Design Service (RDS). Dependent on the rule criteria, the RES could invoke the DAS or MIS for further analysis before issuing the final decision or invoke the Notification-Action Service (NAS) to issue relevant notifications, recommendations, alerts, etc. to the users. The NAS could also send back other action commands to several grid child nodes for executing several data manipulation operations at the local data sources such as insert, delete, update, etc. Furthermore, the analysis process above could be executed to answer the analysis queries issued by other applications. Especially, in case the local data update takes place at the grid child nodes, the analysis process could also be invoked. Whenever the local data update happens, the NAS at local child node sends the “local data updates” message events to the NAS of the Master node. The NAS then invokes the DAS and the Analysis process will be executed.

The invocation between the services described above, in fact, is more complicated in the dynamic Grid environment because of the computational and networking capabilities, and availability of the Grid nodes. Therefore, the Grid services invocation process is strictly monitored by the Resource Broker Service (RRS) and the System Information Service (SIS). These services manage the available resource and find the best-fitting resources for resource allocation and dispatch. The role of Workflow Management Service (WMS) is to execute the complex, highly dynamic workflows involving different grid service instances.

**DYNAMIC GRID SERVICE COLLABORATION**

As previously mentioned, each OGSI based service in GZLDSWH is able to perform an individual task within the whole process. Obviously, these services have to collaborate with each other to fulfill the common purpose of the GZLDSWH system.

The Dynamic Service Control Engine (DSCE) has been developed in GridMiner (Kickinger G. et al., 2003; Kickinger G. & Brezany P., 2004) to control the service execution via Dynamic Service Control Language (DSCL) document. The exact services handles are specified in the DSCL. Users have to know which service factory to use to perform a certain task. Therefore, the services do not need to communicate with each other. The output of the first service serves as the input of the second service, the output of the second one serves as the input of the
third one and so on. No service thus is aware of other existing services and each of the services is able to run completely independently.

The independence of the various services also allows a parallel execution without any communication overhead. This results in an improvement of performance.

However, in our system, due to the requirement of automated event-based reaction, a service must be able to discover, create, bind, and invoke relevant service instances within the Grid environment. Only the execution flows are specified in advance, in which the services are arranged in the specified logical execution order. During the execution time, the services have their autonomy to discover, create, bind, and invoke relevant physical service instances within the Grid environment depending on the context at that time. Consider the Telecommunication Fraud Detection scenario, for example, the Call Detail Records (CDRs) are issued continuously as continuous data streams. The Data Capturing Service (DCS) instance is created in the Master Node and its operations should be executed while the CDRs are still arriving. The DCS instance will invoke the Data Storing Services (DSSs) at several child nodes to store data. However, according to the Grid environment at the runtime, some child nodes are available, the others could be
corrupted or out of resources. The DCS instance thus must be “intelligent” enough to create and invoke the instances at the suitable Grid nodes. If the CDRs data arrives in a burst fashion, the DCS instance has to create many DSS instances to store all data in a timely fashion, otherwise, it should destroy some non-used DSS instances to free the resources. The similar situation happens when the DAS instance at Master node decides to create and invoke the DAS instances at the suitable child nodes. Especially, in case the Rule Evaluation Service (RES) could not issue the final decision (due to ambiguity between Fraud and non-Fraud call), it has to invoke other analysis services or data mining services for further data analysis instead of invoking the Notification/Action service to alarm the users. Therefore, the dynamic service invocation requirement is extremely important in the GZLDSWH.

To our best knowledge, there are 2 possible approaches for the service flow execution: centralized control and distributed control. In the former approach, there is a central service control engine which controls all service executions from the start node to the end node of the workflow. The engine itself is responsible in discovering, creating, binding, invoking, and destroying service instances to follow the logical workflow. Thus the engine must keep the information of the whole workflow and should trace the information of the Grid environment such as grid nodes status, resource availability, workloads etc. to coordinate the services execution. In the later approach, there is no such central engine but each service instance has its own “knowledge” to invoke the next service instances throughout the workflow. It is not necessary for each service to keep information of the whole workflow. Instead, each service needs to keep only part of the workflow metadata relevant to itself such as its direct ancestor and descendant services, and the Grids environment context at its execution time. That information should be passed to the service as parameters as it is invoked by its ancestor services and the service will use such information to invoke the next relevant service instances.

Both of the two approaches have advantages and disadvantages. In the centralized approach, the central service control engine, which could also be realized as a service, copes with the coordination between other services. The other services thus only focus on their specific functionality without taking into account the workflow execution. However, it could be the heavy work-load for the engine service if it processes the high complexity workflow or if the number of service instances increases. The distributed control approach, on the other hand, does not have to deal with the bottle-neck issue. However, it is more complicated to develop the services because each service besides its specific functionality must be realized as an agent-based solution to adapt with flexible service instance invocations. Moreover, the service invocation would also become more complex due to the parameters transferred between the service instances. Further investigation on distributed control approach is out of the scope of this paper. It will be one of our considerations in the future work.

In GridMiner (Kickinger G. et al., 2003; Kickinger G. & Brezany P., 2004), we have developed the Dynamic Service Control Engine (DSCE) which receives the workflow specification document written in DSCL (Dynamic Service Control Language) and executes the workflow by invoking the corresponding Grid service instances specified in the documents. DSCL allows the user to specify variables, workflow structure, operations to be executed, and DSCE will execute the workflow followed by the DSCL document specification. Although the DSCL and DSCE provide some levels of dynamic workflow execution, they still have some limitations. The DSCL does not support branch conditions and loop structures; the DSCE only works with the physical workflow specification document i.e. the document specifies exactly which factory handle URIs should be invoked to create the instances which are usually unknown in advance due to the variant Grid environment. We can improve the DSCL and DSCE in GridMiner to support the new requirements in GZLDSWH. The extended DSCL (Tho N. et al., 2004) will support the condition branches, loop structures as well as allow the references of the service instance handles could be transferred as parameters. The logical workflows are specified with the unknown service instance handles declared as variables. During the execution time, the Re-W riter queries the Resource Broker Service and Information Service to have the relevant dynamic service factory handle references at that time and rewrites the logical workflow to the physical one. The DSCE engine then will invoke
these service instances via the reference variables in the physical workflow. That operation will be repeated at each step of the workflow until the whole process is finished.

**Extended Dynamic Service Control Language**

DSCL is a XML based language allowing the users to specify the workflow of services activities. It contains exactly of two sections:

- The `<variables>` section: all variables must be defined here. The variables could be either the parameters of service calls, or the results of service calls. XML Schema Simple Type, Complex Type and SOAP Arrays Type are supported as variable type.

```xml
<variables>
  <variable name="iAge">
    <value type="int" 25/>
  </variable>
</variables>
```

- The `<composition>` section contains the description of the workflow to be executed. A workflow is comprised of a set of activities which could be classified as "control flow" or "operational". The control flow activities control the execution of the workflow and thus must contain other activities while the operational activities are the atomic activities which perform operations.

```xml
<composition>
  control flow activities
  other control flow activities or operational activities
  operational activities
</composition>
```

DSCL allows the users to specify the workflow structure and define the workflow operations.

**Workflow structure**

Our DSCL supports 4 basic execution styles (Sequential execution, Parallel execution, Condition Branch and Loop) by providing several tags namely `<Sequence>`, `<Parallel>`, `<Condition>`, and `<Loop>` respectively. These tags could be nested to realize the complex workflow composition. Figure 6 states an example of a workflow including all control activities and the respective DSCL document:

![Figure 6. Composite Workflow Example](image)

```xml
<composition>
  <sequence>
    activity1
    <condition>
      cond_var1 = TRUE
      <loop while cond_var2 = TRUE>
        activity2
      </loop>
      cond_var1 = FALSE
      <sequence>
        <parallel>
          activity3
          activity4
        </parallel>
        activity5
      </sequence>
    </condition>
  </sequence>
</composition>
```

**Workflow operations**

Besides the control flow activities, DSCL supports other activities namely operational activities. Operational activities perform operations for interacting with the underlying Grid services such as creating new service instances, destroying instances, invoking operation of services, and querying service data element. DSCL provides respective tags to specify these operational activities: `<createService>`, `<destroyService>`, `<invoke>`, and `<querySDE>`. The operational activity could not contain other activities and must have the mandatory attribute namely activityID which is of type DTD. This attribute is necessary for the workflow engine to identify the activity.

The major difference between Grid and common Web services is the fact that the Grid service could...
be either persistent or transient. The persistent service is created and available if its container is running. In contrast, the transient one is created and invoked as and when required and destroyed afterwards. The transient service is always created by its Factory service. The following information is necessary to create a new service instance:

1. The location of the factory service
2. Additional service parameters
3. A virtual instance name of the newly created instance

```xml
<createService activityID="Act1"
factory-gsh="http://url/serviceFactory"
instance_name="newInstance1"/>
```

In GZLDSWH, there is a situation when we have more than equivalent factory services at different grid nodes at the same time. For example, the Data Storing services located at several child nodes when we need to store data stream. In such situation and in other cases when the Grid environment changes rapidly, the Resource Broker Service decides which Service Factory should be executed to create a new instance according to the availability and resource capacity of the different Grid nodes. DSCL provides the dynamic service creation by allowing Grid service handle references transferred via variables. It is also possible to create the service instance with user defined parameters via `<parameter>` tag.

```xml
<variables>
  <variable name="factgsh">
    <!-- Default value of the factory service handle -->
    <value type="string" http:url/serviceFactory /></value>
  </variable>
</variables>
```

Some of operations do not return the results; but store them into so called service data elements. To allow querying the contents of these elements, DSCL provides the tag `<querySDE>`. This operation requires the reference to Grid service instance and the name of the required service data element (stored in attribute `sdName`).

```xml
<querySDE activityID="act1"
  instance_name="instance01"
  sdName="value"
  <result variable="var02"/>
</querySDE>
```
Dynamic Workflow Management Service

In GridMiner project (Kickinger G. & Brezany P., 2004), we have developed an engine service, called the Dynamic Service Control Engine (DSCE), which processes DSCL documents and controls the service execution in both interactive and batch modes. It provides some interesting features such as (a) independent processing (without any interaction of the user) of a workflow described in DSCL, (b) the provision of all intermediate results from the services involved, (c) the possibility for a user to stop, cancel or resume a workflow and (d) the possibility to change workflow at run time (by stopping the engine, changing the DSCL document, and restarting engine again).

Figure 7. Conceptual Architecture of DSCE

Figure 7 describes the Conceptual Architecture of DSCE (Kickinger G. & Brezany P., 2004). The engine is implemented as a stateful, transient OGSI Grid service and has several structured layers. The “top” layer is the Interface layer which provides essential operations to control the engine. The Factory interface allows users to create a new DSCE instance for a specific DSCL document via operation CreateService (DSCLDocument dscl). The DSCE engine instance now will be created and its state will change within its life cycle according to user interactions and the activities execution results. The possible states could be empty, initialised, running, stopping, waiting, finishing, or failure. The Service interface provides interactive control operations such as changeWorkflow(), start(), stop(), resume() as well as several service data elements containing information about the DSCL document, Workflow state and activity state.

The “middle” DSC Engine layer covers the main functionality of DSCE. It controls the whole workflow execution by controlling the execution of activities specified by the DSCL document. First, the DSCL workflow description is parsed, then the “network of activities”, an internal model of the workflow, is constructed before processing the activities. Such a network of activities describes the dependency between the activities. Each activity could have succeeding and preceding activities. Succeeding activities are executed right after the execution of actual activity is finished. If an activity has more than one successor, all of them will be executed in parallel after the actual activity is finished. Similarly, the activity could not be started until all of its preceeding ones are finished. This could happen in some situations like loop or parallel execution. Several internal operations are provided in this layer for managing the workflow such as start(), stop(), resume(), reset(), setDSCLWorkflow() etc. as well as some operations for controlling the activities such as startActivity(), EndActivity(), CreateInstanceActivity(), DestroyInstanceActivity(), InvokeActivity(), QuerySDE-Activity(), startNextActivites(), wait-ForPrevious-Activities(), etc. The necessary parameters of all underlying services are also prepared at this layer.

Normally, when a Grid service is developed and implemented, additional stub and proxy classes are generated to hide the complexity of communication between the client and the service. This approach is very common and practically used in all distributed object systems like CORBA, Java RMI, and Web Service. To benefit from this approach, the required services or remote objects must be known at the
Figure 8. The Extended Service Control Engine and corresponding DSCL documents

```
<dscl:variables>
    <variable name="dataArrive"> TRUE </variable>
    <variable name="dataNotClean"> TRUE </variable>
    <variable name="dsfgh"> // unknown value
    <variable name="dssfgsh"> // unknown value
    <variable name="dsfgh1"> // unknown value
    <variable name="dsfgh2"> // unknown value
    <variable name="tcsfgsh"> http://MasterDCSFactory </variable>
    <variable name="dsfgh"> http://MasterDCSF
    <variable name="tcsfgsh"> http://MasterDCSF</variable>
    <variable name="tcsfgsh"> http://MasterDCSF</variable>
    </dscl:variables>

<dscl:variables>
    <variable name="dataArrive"> TRUE </variable>
    <variable name="dataNotClean"> TRUE </variable>
    <variable name="tcsfgsh"> http://MasterDCSFactory </variable>
    <variable name="dsfgh"> http://MasterDCSF</variable>
    <variable name="tcsfgsh"> http://MasterDCSF</variable>
    <variable name="tcsfgsh"> http://MasterDCSF</variable>
    // other variables
</dscl:variables>

<composition>
    <sequence>
        <loop while dataArrive = TRUE>
            invoke DCS
        </loop>
        <condition> dataNotClean = TRUE
            invoke DES
        </condition>
        <sequence>
            invoke several DSSs
        </sequence>
    </sequence>
</composition>
```

compilation time. However, this requirement is not satisfied in DSCE because DSCE receives a DSCL workflow description document and shall be able to communicate with all services specified within that DSCL. The “lowest” layer namely Dynamic Grid Service Invocation (DGSI), is composed of the DGS Invocation and Dynamic Invoker. It provides classes which allow accessing Grid services and their operations without using common stubs/proxy approach. The Dynamic Invoker, the lowest layer, provides the possibility to invoke any operation on any underlying Grid service. It uses much of ApacheAxis (The Axis Project, 2003), and SOAP engine which are based of GT3 toolkit. Dynamic Invoker translates an operation invocation into a SOAP1.1 message and sends it to the corresponding service to invoke specified operations. It provides all necessary marshalling and un-marshalling of arguments, first by fetching the WSDL of the corresponding Grid service (via its handle GSH), and then setting service port type via

```java
setPortType(String port-Name),
```

setting operation via `setOperation (String operation Name)` , setting parameters of the operation via `setParameters (Object[] params)`. All of information is used to construct essential SOAP operation call. Finally `invoke()` executes the operation by sending that SOAP message to correspond services. At higher layer, the DGS Invocation provides the classes to use stub-less operation invocation and to access the functionality of the GT3. It provides three classes namely `DGSIService, DGSIFactory, DGSIF-Listener` allowing the workflow engine to handle its underlying services such as creation and destruction of Grid service instance, invocation of operations, querying of service data element and synchronization of asynchronous service calls.

DSCE suits well in GridMiner where the interaction role of user is important. The engine operates based on the "physical" DSCL document specified by the user, i.e. it only works with the DSCL that specifies exactly the service handles. It
does not accept the “logical” workflow which only specifies the logical name of the required service. In GZLDWSH, we sometimes do not know in advance which service factory should be executed to create a new instance. Instead, the decision should depend on the runtime environment. Besides, because of the “automated event-based reaction” feature of GZLDWSH, a higher level of automation in service invocation engine is required. Therefore, the Dynamic Workflow Management Service in GZLDWSH extends the DSCE with the automatic Workflow Re-writer ability. Now, the WMS Service will accept the logical DSCL, parse it and find out logical services i.e. services which do not have the exact physical factory handle. It then queries the Resource Broker Service to have the relevant physical service factory handle and then re-write the DSCL with the new factory handle value. It finally passes the re-write DSCL to the DSCE engine to invoke the services.

Figure 8(a) describes the extended Dynamic Service Control Engine (EDSCE) based on that DSCE engine. This EDSCE engine accepts the logical workflow in Figure 8(b), parses it and converts to the physical executable workflow as depicted in Figure 8(c). The significant difference is the values of the parameters that have been supplied after the Re-Writer queries the Resource Broker Services.

THE OLAP CUBE MANAGEMENT SERVICE

As mentioned previously in the overview section, our approach in GZLDWSH is to store all streaming data into Grid nodes, and build OLAP cubes from these Grid-based sources prior to executing analytical queries that evaluate the rules. Following this approach, we have to implement the OLAP Cube Management Service that manages the creation, updation and querying of the associated cube portions distributed over the Grid nodes. The kernel part of this service is the OLAP engine. The first prototype of this engine (Fiser B. et al., 2004) has been already implemented in Java.

The OLAP data cube structure consists of an increasing number of chunks, which again consists of a fixed maximum number of measures. A measure is the smallest unit of the cube, one atomic element, and it actually contains just a numeric value. The chunk is a part of the whole cube; it has the same dimensionality like the cube but collects aggregation data at one grid node. The chunk contains the measures associated with a number of positions of each dimension. Because the amount of memory used by the whole cube usually will be much higher than a system may provide, each chunk offers methods for storing and loading its data onto and from the disk storage. Thus, always only a limited number of chunks is kept within memory at the same time. Storing and loading targeted chunks is called chunk swapping and is a subsystem of the data cube structure implementation. This is similar to paging in modern operating systems with the distinction that our chunks may grow up to a specific size, hence, the memory resident chunk location table, which is a list of chunks currently resident in memory, varies in size. This is because the aggregation results are also stored within the same data cube.

Special indexing structures and paging mechanisms are necessary to manage the Grid-based OLAP cubes. The index database contains the literal positions, the meta-information, of each dimension and maps unique integer values to position indexes within each dimension. Furthermore, it provides methods for the linearization of multi-dimensional position indexes used for addressing specific measures of the OLAP cube. Several methods are available, e.g. hashing, bit encoded sparse structure (BESS), binary trees or others. In order to deal with a huge number of tuples, we need an algorithm, which on the one hand is fast, and on the other hand, is not limited to some upper boundary. This is necessary to avoid multiple scans of the source data and allows insertion of measure aggregation after cube construction. The method, called Dynamic bit encoding (DBE) (Fiser B. et al., 2004) was developed for indexing the bit maps of OLAP chunks. DBE is based on BESS with the difference that the bits used for each dimension are kept within bit masks which are extensible and mutually exclusive to each other from a binary point of view. The position indices are processed by the OR operation using these masks, which results in a linear measure address, called the global index.
Figure 9. OLAP engine architecture

Figure 9 describes the system architecture of the OLAP engine prototype. The cube Construction reads the tuples one after the other, passes over the items to the index database, retrieves their global index and then passes the (raw) measure and its associated global index to the data cube structure. The Querying functional block is some kind of highly sophisticated, recursively nested loops for aggregation of measures. Because the number of computational operations of nested aggregations depends on the size of the dimensions and thereby on the order in which dimensions are aggregated, the engine uses a kind of query plan optimization to select dimensions in a "good" way. The procedure of dimension selection is done by traversing a tree which in the literature usually is called the query lattice. The task of aggregation is realized sequentially by loading one chunk after the other and aggregating them one by one. To avoid repeated computation of same aggregates, they are also stored in the cube structure as if they were raw measures and also get index entries within the index database within appropriate dimensions.

Connection handling is the network interface, which allows user interactions with the system. A typical workflow of the system usage is as follows. After start-up, the index tables and the base cube are constructed. This is done upon loading and parsing a structured or semi-structured text file representing database tuples. It is called the origin input stream (see Figure 9). To each position, a unique index value is assigned and this assignment is kept within the index database. Then all index values from the tuple are merged together using the DBE algorithm. The encoded global index is used to uniquely locate and store the measure within the cube. After the step of tuples import, the server opens a listening TCP socket and accepts client connections. A simple command language was defined (Fiser B. et al., 2004) for communication between server and client. This is called the control input (output) stream. A client now is able to submit queries. The server supports concurrent sessions, which allows multiple users to login concurrently.

**KNOWLEDGE BASED RULE DESIGN AND EVALUATION MECHANISM**

The Knowledge Based Rule is the “brain” of the GZLDSWH and controls how the system reacts automatically to the continuously arriving events based on the complex incremental multi-dimensional analysis of the collected OLAP data. The rules follow the basic Event-Condition-Action (ECA) rule structure, but carry out the complex OLAP analysis instead of evaluating the simple conditions as in ECA rules in OLTP. It is necessary to design a model for maintenance of the Knowledge Base rules that allows users to insert, update, replace or delete the rules easily. If possible, the rules should be managed in a consistent manner with the option of checking the validation of the rule, avoiding rule conflicts, and maintaining the consistency among the rules.

The mechanism to evaluate the rule is also a challenge when the events come from different sources within the Grid environment. There are several causes that can trigger a rule. It could be the temporal time events generated from the scheduling service, explicit invocation from other service or auto-triggered when the Data Service Element (DSE) in another service exists. Right now, we just use a simple XML-based file to manage the rules. The Rule Evaluation Service is explicitly invoked by the Workflow Management Service.

```xml
<ReactiveRule name = "Mobile Fraud Analysis">
<variables>
  <variable name="average_call_length"/>
</variables>
</ReactiveRule>
```
Figure 10. Mobile Phone Fraud Detection Rule

Figure 10 describes a Fraud Detection Rule which monitors the international mobile call. This rule will be triggered when the current international call length to Vietnam of a certain customer is over 30 minutes. It will check whether the average call length of this customer to other countries in South East Asia such as Thailand, Malaysia, Indonesia within the last 3 hours exceeds twice the average call length to Asia for this customer. If so, the Fraudulent Call alarm will be issued and sent to the customer.

RELIABILITY AND EFFICIENCY

In our architecture there are two options of how incoming data streams are assigned to nodes in a Grid. First, a master node can select which node an incoming data stream will be stored on. Second, any node in the Grid may accept an incoming data stream. The node can choose to handover the data stream to the master node at any point in time.

Grids are characterized by a large number of cooperating nodes. The rationale is that storing of multiple data streams can be handled in parallel and is thus more efficient. Amdahl’s law (Gene A., 1967) describes the speedup of programs if the performance of some parts (but not all) can be improved. A special case of this improvement is parallel processing. If parts of the code can be executed in parallel, the overall performance will be better.

\[ \frac{1}{S + \frac{1-S}{N}} \]
shows that speedup depends on how much must be executed sequentially (S in the range of 0.0 to 1.0).

As the size of the grid (N nodes) increases, reliability of nodes becomes an issue. Even though the mean time to failure (MTTF) of individual nodes decreased over the last years, the size of grids grows faster so that the MTTF of the system decreases if one requires all nodes to be online. Clearly, in a large grid with N nodes, the setup usually requires only A nodes (A<N) to be available for the grid to work.

If a node receives an incoming data stream and then fails, parts of the data stream might be lost. To avoid inconsistent states, the failed node and possibly some neighboring nodes will need to roll back to the last checkpoint.

Following Elnozahy’s models (Elmootazbellah N., James P., 2004) we look at specific requirements of data streams and how system parameters need to be modified to achieve the primary goal of increasing performance. Each failure causes additional work to be performed; useful work is work that a system performs that will not be lost by a failure. Obviously, the goal is to maximize the ration U of useful work / total work. When serial parts are low then increasing the number of active nodes A to values near N increases U (Elmootazbellah N., James P., 2004). Building on Elnozahy’s results, simulations show that checkpoint intervals for grids with fewer than 4000 nodes, should be greater than 20 minutes. For growing numbers of N the optimal (maximizing U) interval to set checkpoints is approximately the checkpoint latency, assuming a checkpoint latency of five minutes.

To minimize overhead of creating checkpoints it is advisable to reduce the length of serial parts in the process of storing data streams. This can be achieved by distributing the stream to several nodes: for instance, 32 KBytes to the first node, the subsequent 32 KBytes to the next node, etc. Assuming a data
rate of 100 KBit/s the sequential duration of a 32 KByte block is \( \frac{32 \times 1024 \times 8}{1024 \times 100} = 2.56 \) sec.

By adjusting the block length of the stream the overhead of setting checkpoints can be adjusted. Shorter blocks will reduce the overhead but will increase fragmentation of the stream. Depending on the requirements of retrieving and analyzing the data stream a decision concerning the tradeoff checkpoint overhead versus data fragmentation has to be made.

**IMPLEMENTATION**

In this section, we will describe our ongoing prototype implementation and some experimental performance results. So far, we have developed the prototypes for the Dynamic Service Control Engine (DSCE) and the Sequential OLAP Cube Engine.

**Dynamic Service Control Engine Prototype**

Figure 11 describes the class architecture of the DSCE prototype which follows the conceptual architecture mentioned earlier. It is implemented as a stateful and transient OGSA Grid service.

DSCE is a workflow enactment engine, which executes a workflow of OGSA Grid Services described by the Dynamic Service Control Language (DSCL). The engine is always in a particular state. Dependent on its state, it could accept the client’s command, executes the command and changes the state respectively. This state tells the client about what the engine is doing at the moment and if it is ready to accept certain commands. The state diagram of the DSCE engine is depicted in Figure 12.
After being created as a service instance, the DSCE engine is in *empty* state. If a client sends a new DSCL workflow description and the engine accepts it, its state will change to *initialized*. The state will change to *running* if the client starts the workflow and the change of DSCL workflow description is not possible thru `changeWorkflow()` command. If the client need to change the DSCL description, the engine execution has to be stopped first and its state will change to *stopping* (during the processing stopping the activities) and *waiting* (when all stated activities have been returned) respectively. The engine could finish without error (*finished* state) or stop the execution with error report (*failure* state).

To execute the performance tests, we define a minimal sub-workflow with no computational cost. This is used as the smallest item throughout all test runs. The sub-workflow consists of the following sequence:

1. Create new `TestGridService` instance
2. Invoke the operation `getFloatValue()`
3. Invoke the operation `setFloatValue(float)`
4. Query the service data element
5. Destroy previously created `TestGridService` instance

To test the workflow engine, 24 different DSCL documents are used (pt1.xml, pt2.xml,..., pt24.xml). DSCL document pt1.xml contains exactly one of the previously defined sub-workflow, DSCL document pt2.xml contains two successive sub-workflows and so on. Finally, DSCL document pt24.xml contains 24 successive sub-workflows, this result in $24 \times 5 = 120$ activities.

A client application (*DSCE-Client*) is implemented to invoke the DSCE service instance, one for each DSCL documents. To determine the overhead of the DSCE engine compared to a direct execution of the workflow, another client has been implemented which executes the sub-workflows without usage of DSCE and DSCL (*DirectClient*).

In Figure 13, we can see the execution time of the overall workflow between the two methods: invoke DSCE engine or direct execution of the workflow.
Figure 13. Comparison of overall workflow execution time.

**Sequential OLAP Cube Engine**

Our prototype implementation of the OLAP engine was tested with three input files - tab separated text files - of different sizes. The test files were structured in a way so that each column represents the values for a dimension, whereby the first column contained the measure values for the cube. Using these input files, we have built three different OLAP cubes with four dimensions, which mean that each input file contained data in five columns. The following table gives an overview of the characteristics of our test files.

<table>
<thead>
<tr>
<th></th>
<th>Input I</th>
<th>Input II</th>
<th>Input III</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOL</td>
<td>2000</td>
<td>8000</td>
<td>20000</td>
</tr>
<tr>
<td>NODV in column I</td>
<td>100</td>
<td>300</td>
<td>6000</td>
</tr>
<tr>
<td>NODV in column II</td>
<td>10</td>
<td>15</td>
<td>25</td>
</tr>
<tr>
<td>NODV in column III</td>
<td>45</td>
<td>86</td>
<td>125</td>
</tr>
<tr>
<td>NODV in column IV</td>
<td>2000</td>
<td>8000</td>
<td>20000</td>
</tr>
<tr>
<td>BPCS</td>
<td>90000000</td>
<td>309600000</td>
<td>3750000000</td>
</tr>
</tbody>
</table>

Table 1. Characteristics of the test files used in the test scenarios.

- **NOL**..............Number of lines
- **NODV**.............Number of distinct values
- **BPCS**..............Biggest possible cube size

\[(\text{BPCS} = (\text{NODV} - 1) \times (\text{NODV} - 2) \times (\text{NODV} - 3) \times (\text{NODV} - 4))\]

We have treated each test session – which uses a different input file - as a new OLAP scenario and sent the same queries to the OLAP cube in order to provide an overview of the querying performance relative to the increasing cube size. Each query is represented in a form like \([\text{ANY, ANY, ANY, 0}]\) whereby the term \text{ANY} indicates that the cube shall be aggregated along this dimension. This means that the costs for a query increases with the increasing number of distinct values in a dimension, when the query value is set to “\text{ANY}”. Figure 14 demonstrates the first performance results, which are more than satisfying for a prototypical sequential OLAP engine. In our future research effort, we plan to investigate the architecture of a parallel and distributed OLAP engine.

**CONCLUSIONS AND FUTURE WORK**

In this paper we have presented a framework of building the Grid-based Zero-Latency Data Warehouse (GZLDSWH) system for continuous data streams processing and analysis. The GZLDSWH system is built upon the set of OGSI/GT3-based services. Following the pre-defined reactive Rule-based Metadata specified by the user, the system can react automatically to continuous data streams in near real time. An adaptive service interaction mechanism is used for...
the flexible service collaboration. The system then can execute some relevant actions at the data sources or send significant awareness such as alarms, notifications, recommendations, etc. to the user.

Our proposed GZLDSWH system is currently an ongoing research and some of the other issues require further investigation. Major focus of future research should be on solving several significant services within the GZLDSWH such as Grid Resource Allocation and Scheduling, Heterogeneous Data Mediation and Integration, Grid Data Replica Synchronization (Goel S., Sharda H., & Taniar D, 2005), Data Cube Construction and Management, Rule-based Metadata Construction, Embedding and Evaluation. Other open issues such as Performance Efficiency in building Data Warehouse on the fly, Distributed Service Control Management, Stream-based Distributed Processing, Heterogeneous Stream, Availability and Security should also be considered. Besides that both Grid and Data Stream processing technologies are young and still evolving. The Semantic Grid (Roure D., Jennings N., & Shadbolt N., 2003) and Grids services have the role similar to Semantic Web and Web services. Recently, WS-Resource Framework & WS-Notification (Globus Alliance, IBM, & HP, 2004) proposals have been announced as an evolution of OGSI with the purpose of effective integration of Grids and Web services standards. The work presented here is closely related to OGSA/OGSI so it has to adapt with the WS-Resource Framework with suitable modifications. The distributed control of service discovery, creation, invocation and destroying will be considered as an alternative of collaboration model. The orchestration of Grids or Web services, another approach for solving the workflow problem, should also be further investigated.

ACKNOWLEDGEMENT

This work has been partly supported by a Technology Grant South East Asia (No. 322/1/EZA/2002) of the Austrian Council Research and Technology and the ASEA-UNINET.

We are very indebted to Umut Onan and Günter Kickinger for their contribution concerning the implementation aspects. Thanks to Shuaib Karim and Khalid Latif for their English revision efforts.

REFERENCE


Tho Manh Nguyen received his Master degree in Computer Science at HoChiMinh City University of Technology, Vietnam, in 1998. Since October 2002, he is Research Assistant at the Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria. His research areas cover Data Warehousing, Knowledge Discovery, Continuous Data Streams, and Grid Computing.

Peter Brezany is a professor at the Institute for Software Science, University of Vienna, Austria. He received his PhD in Computer Science in 1980 from the Slovak Technical University Bratislava, Slovakia. He began research in 1976 on the design of parallel programming languages and their compilers. Since 1990 he has worked on automatic parallelization of scientific and engineering applications for distributed-memory systems, parallel input/output support for high performance computing, and large-scale parallel and distributed data mining. His current research focus is knowledge discovery and data management on Computational Grids.

A. Min Tjoa is a full professor and the director of the Institute of Software Technology and Interactive Systems at the Vienna University of Technology. He received his PhD in Engineering from the University Linz, Austria. His research interests include Semantic Web, e-Commerce, and advanced and scalable data management and data analysis solutions for management information systems and decision support. He is member of the Technical Committee for Information Systems of the International Federation for Information Processing. He is the chair person of the 2005 DAWAK Conference (7th International Conference on Data Warehousing and Knowledge Discovery).

Edgar R. Weippl holds a MS and a Ph.D. in Computer Science, and an MA in Business Administration. He has presented tutorials at various conferences, and actively participates in the scientific community. His general areas of interest are security and e-learning. Specifically, his work focuses on security in e-learning.