A Communication Infrastructure for a Distributed RDBMS

Michael Stillger, Dieter Scheffner, Johann-Christoph Freytag

Computer Science Department,
Humboldt University at Berlin, Germany
[stillger,scheffne,freytag]@dbis.informatik.hu-berlin.de

Abstract. We present the concept and implementation of a communication infrastructure for a distributed database system referring to the agent-based database query evaluation system AQuES. Within this model we use system components that build a federated multi agent system (MAS). We present those parts of the message transport layer that provide an "easy to handle", scalable architecture based on the "plug & play" building block principle. Furthermore, we present a generic dialog manager that enables each agent to communicate in multiple concurrent threads of execution. Based on this concept, AQuES agents keep track of a complex evaluation environment in a dynamic, multi-query scenario.

1 Introduction

Focusing on runtime query optimization for a parallel and distributed execution environment, the AQuES [6] system was designed as a multi agent system to efficiently answer SQL queries in a distributed environment. We assume query execution to take place in an open system that is subject to changing workloads and a varying agent communities competing with ordinary multi-user tasks for resources at each node and at any point in time. Dynamic optimzation is carried out to compensate unpredictable resource parameters in such environment. This overall complexity is characterized not only by data streams to be distributed in a flexible way, but also by the message flow to be managed. All components of AQuES communicate via the KQML Agent Communication Language for executing and dynamically optimizing queries, thus producing unpredictable communication flow and data flow. For this reason, a uniform, flexible, and efficient communication infrastructure is necessary.

In our AQuES system, components are computational entities that have properties like reactivity, autonomy, adaptability and goal oriented behavior [5], thus they are agents. A set of interacting software agents that cooperate in solving a global task using a facilitator is called a multi agent system (MAS). Agent communication is based on exchanging messages (KQML) [8]. We extend the facilitator concept of MAS towards a set of communicating federations, with each facilitator representing all associated agents to the rest of the system [1]. The facilitator mimics a single complex agent to other federations by integrating all services for building up a federation. Figure 1 shows such a multi federation architecture

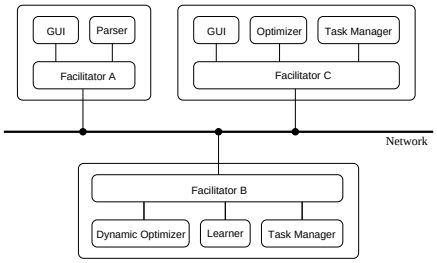

Fig. 1. Federation Architecture

with one facilitator and federation per node. Each AQuES federation might consist of one or more component agents: a graphical user interface (GUI), a parser, a static optimizer producing an optimal execution plan, a task manager, a dynamic optimizer, a learner monitoring the resources, a facilitator managing the communication flow among local components, and a communication component managing the connections to remote AQuES federations. Each federation is responsible for a local database partition and serves as a cooperative entity for the global query execution task. We now present the communication infrastructure that supports the *modular concept* of the system in a *scalable* manner for an arbitrary number of components. For more details, we refer to our technical report [7].

2 The Communication Architecture

In the AQuES message transport layer we distinguish between *intra-federation communication*, i.e., agents' communication within one federation, and *inter-federation communication*, i.e., communication among federations over the network. The following components provide the core for the AQuES communication infrastructure: Message Buffer, active Agent Adapter, Agent Plug-In, Facilitator/Router component and Network Communication Component.

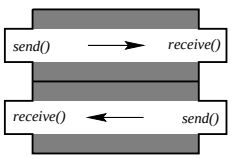

Fig. 2. Message Buffer Elem.

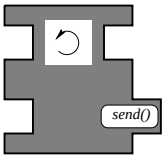

Fig. 3. Agent Adapter

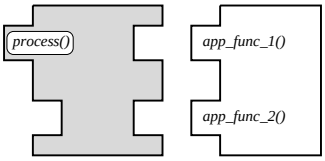

Fig. 4. Agent Plug-In

The Message Buffer. This module forms the basis for intraprocess communication, i.e., it supports the exchange of messages between producers and consumers. Each agent is both a producer and a consumer of messages. For this reason, we use a Message Buffer Element (MBE) with two FIFO message queues (Fig. 2) for each instance of a component. MBEs are scalable in their sizes and each element provides two *send()*-functions and two *receive()*-functions. For processing on the same operation system, we implemented queues in shared memory, thus an MBE keeps only pointers to messages, making the transport layer a very fast medium.

The Agent Adapter. The Agent Adapter (Fig. 3) is the connecting link between the Agent Plug-In and the Message Buffer. It hides the access to a particular MBE by providing more general/global counterparts of *send()* and *re-*

ceive(). In addition, the Agent Adapter supplies a thread-based execution skeleton. Whithin any agent application can be executed with low overhead for context switches and fast concurrent message passing. The loop actively receives KQML messages and let the Agent Plug-In (Fig. 4) process (by *process()*) the messages on behalf of a particular agent.

The Agent Plug-In. The Agent Plug-In maps the application's functionality to a single single *process()* handle and is the connecting link between the application's internals and the Agent Adapter. Invoking the process method causes actions, namely application function calls and sending answer messages according to the agent's dialog protocol. Moreover, the Agent Plug-In supports the addressing of messages and the dialog management (see Sec. 3). The Agent Plug-In and the Agent Adapter give an agent its "reactive" behavior.

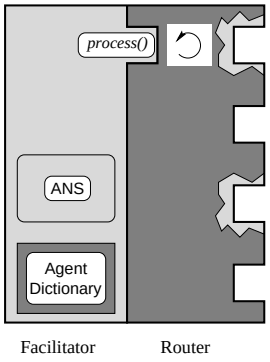

Fig. 5. Router/Facilitator for sending messages [1].

The Router. Unlike the KQML Internal Architecture [8], we only use a single router thread for the message exchange among agents on one machine, including the facilitator. The router directly receives the messages from the MBE's of all local agents. Furthermore, the router invokes the facilitator as its application for each message received, thus the facilitator runs within the router's thread (Fig. 5).

The Facilitator. The facilitator manages the dynamic (un)registration of local agents, global error handling and address resolution by means of an *Agent Dictionary*. Unlike ordinary agents, the facilitator uses *direct communication* with its MBE

Network Communication Component (NCC). The NCC extends the message transport layer by enabling inter-federation communication. The NCC is designed to appear like any other component agent in the local federation, thus providing a single message buffer interface for all remote communication links. We implemented the *Network Access* component as part of the NCC, using a freely available KAPI [4] software package supporting socket connectivity as an alternative to CORBA [3]. The NCC is also thread-driven. Unlike the Agent Adapter listening to the receive-port of the MBE , the NCC listens to the Network Access component for incoming messages that are buffered in a MBE queue while outgoing messages are handled through to the correct socket connection, instantly. Being a proxy to all local agents, the NCC has an agent-like appearance to the local federation provinding communication links to remote federations of the system.

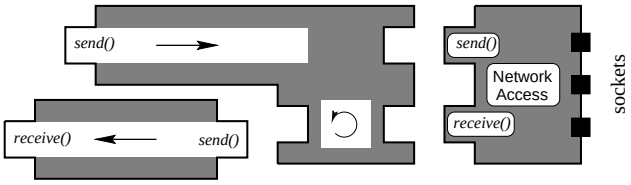

Fig. 6. Network Communication Component

3 Dialog Management

We also include the dialog management (DM) as part of the Agent Plug-In library. The DM and the concept of a *dialog* provide support for choosing the appropriate answer messages, for keeping a record which messages were sent, and for enabling asynchronious concurrent interaction with multiple partners. We explain the dialog mechanism by describing the protocol, the dialog and the dialog manager.

- Protocol: *A protocol is a state transition matrix. For each state it defines one or multiple performatives (messages identified by their performative) that the agent can accept. It defines a transition T(S, P) → (S', A) where S is the current state, P is the received performative, S' is the new state, and A is the agents action that is executed.*
  The received message is a parameter of the action function. If necessary, any reply message will be sent from inside the action function A. Upon returning from the action A the agent is in the new dialog state S' waiting for a new message. Note that the performative of the received message determines the state among multiple successor states in the protocol.
- Dialog: *A dialog D is a four tuple (I, K, P, S) where I identifies the initiating agent of this dialog. K is the dialog identifier that was created at the begin of the dialog. P is the protocol that was chosen for this dialog and S is the current state of the agent in this chosen protocol.*
- Dialog Manager: *The dialog manager of an agent consists of a set of protocols and a set of open dialogs. It provides two functions: answer and issue.*
  - *answer(msg)* is used to react to an incomming message. The dialog manager finds the appropriate dialog from the list of open dialogs identified by K. It maps the current state S of this dialog together with the received performative into a new state S' and executes the associated action A.
  - *issue(new msg)* is used by an agent to create a new dialog. The local dialog manager chooses a new protocol according to the given performative and creates a new open dialog entry and dialog id. The message is than sent out and the dialog manager of the receiving agent also opens a new dialog with the appropriate answer protocoll (see *answer*).

Table 1 shows the simplified transition matrix of a protocol for a facilitator to coordinate a query answering process. The dialog is started by a graphical user interface that sends an SQL query to the facilitator. Each individual agent taking part in this protocol can start a new dialog in order to achieve its subgoal. For instance, any task manager involved might contact a dynamic optimizer agent or another task manager to resolve runtime problems (which is not shown here). For example, Table 1 shows an agent in state 3. It can accept a stream of messages from the task manager containing the result of a query (Tell), the last page of the result stream (Eos), or an error message (Sorry) indicating that the evaluation of the query plan failed. By providing a generic dialog manager in the *Agent Plug-In* block of the infrastructure, we greatly simplify the creation and integration of new component agents into the overall system. We only need to specify the protocols of an agent as well as its corresponding action functions.

State	*Message*	*Action*	*New State*	*Comment*
0	Evaluate	::start_evaluate()	1	//receive SQL
1	Tell	::react_parser_reply()	2	//ask optimizer
1	Sorry	::sorry_from_parser()	Finish	// SQL error
2	Tell	::send_to_taskmanager()	3	// evaluate QEP
3	Tell	::forward_stream()	3	// send result pages
3	Eos	::end_of_result()	Finish	// send last tuple and commit
3	Sorry	::sorry_from_tm()	Finish	// abort

Table 1. Dialog Protocol Matrix

4 Conclusion

The agent paradigm and MAS are appealing approaches to handle the complexity of distributed and parallel database systems including the changes of their dynamic execution environment. Within our AQuES system we extended the MAS concept towards a multi-federation concept using message passing to cope with the unpredictable data flow that can occur in dynamic query execution scenarios. We introduced an *efficient* communication infrastructure suitable to support the *modular concept* of AQuES and to provide a *scalable* architecture for any number of components. Building blocks of the message transport layer were designed to smoothly integrate intra- and inter-federation communication and to implement a generic agent execution framework. Moreover, we presented a dialog infrastructure that enables asynchronous and concurrent communication flow among agents.

References

1. Genesereth, M. R. and Singh, N. P. and Syed, M. A Distributed and Anonymous Knowledge Sharing Approach to Software Interoperation. In *International Journal of Cooperative Information Systems*, volume 4, pages 339–367, 1995.
2. G. Graefe. Volcano - An Extensible and Parallel Query Evaluation System. In *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, volume 6(1), pages 120–135, February 1994.
3. The Object Management Group. *CORBA/IIOP 2.2 Specification(98-7-01)*. OMG, `http://www.omg.org/`, 1998.
4. Jay Weber, EIT. `ftp.eit.com/pub/shade/kapi*` see also: `http://hitchhiker.space.lockheed.com/aic/shade/software/KAPI`.
5. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. In *The Knowledge Engineering Review*, 10(2):115–152, 1995.
6. M. Stillger, J. K. Obermaier, and J.-C. Freytag. AQuES: An Agent-based Query Evaluation System. In *Proc. Int'l. Conf. on Cooperative Information Systems*, Charleston, SC, USA, June 1997.
7. Michael Stillger, Dieter Scheffner, and Johann-Christoph Freytag. A Communication Infrastructure for a Distributed RDBMS. Informatik Bericht 137, Computer Science Department, Humboldt University at Berlin, Berlin, Germany, 2000.
8. Tim Finin and Richard Fritzson, Don McKay and Robin McEntire. KQML as an Agent Communication Language. In *The Proc. of 3'rd Int'l Conf. on Information and Knowledge Management (CIKM'94)*. ACM Press, November 1994.
9. Yun Wang. DB2 Query Parallelism: Staging and Implementation. In *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, Zurich, Switzerland*, pages 686–691. Morgan Kaufmann, 1995.