

Distributed Query Adaptation and Its Trade-offs

Henrique Paques
Georgia Inst. of Technology
College of Computing
paques@cc.gatech.edu

Ling Liu
Georgia Inst. of Technology
College of Computing
lingliu@cc.gatech.edu

Calton Pu
Georgia Inst. of Technology
College of Computing
calton@cc.gatech.edu

ABSTRACT

Adaptive query processing in large distributed systems has seen increasing importance due to the rising environmental fluctuations in a growing Internet. We describe *Ginga*, an adaptive query processing engine that combines proactive (compile-time) alternative query plan generation with reactive (run-time) monitoring of network delays. The core of *Ginga* approach is the notion of adaptation space and mechanisms for coordinating and integrating different kinds of query adaptation. An adaptation space consists of a set of adaptation triggers and a set of adaptation cases associated with the triggers. Each adaptation case describes a specific adaptation opportunity of the query execution when changes to the runtime environment are detected. Our experimental results show that *Ginga* query adaptation can achieve significant performance improvements (up to 40% of response time gain) for processing distributed queries over the Internet.

Categories and Subject Descriptors

H.2.4 [Systems]: Query Processing; H.1.0 [Models and Principles]: General.

General Terms

Design, Reliability, Experimentation, Performance.

Keywords

Query Adaptation, Distributed Query Processing.

1. INTRODUCTION

In contrast to the closed world assumption made by most of conventional databases, advanced Internet applications operate in an open environment that changes dynamically. In particular, network bandwidth and latency fluctuate quite unpredictably. The problem is aggravated for long-running *ad hoc* queries and continual queries [12] that are executed repeatedly, since significant changes in runtime environment

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC 2003, Melbourne, Florida, USA.

Copyright 2003 ACM 1-58113-624-2/03/03 ...\$5.00.

can make any static query plan sub-optimal over time. Adaptive query processing has been recognized as an important problem with significant research activity (see Section 6).

Most of the previously proposed approaches to adaptive query processing can be classified [1] as either proactive (i.e., calculate the adaptation alternatives at compile time) or reactive (i.e., find appropriate adaptations at run-time according to environmental changes). In our adaptive query processing engine, called *Ginga*, we combine proactive alternative query plan generation with reactive monitoring and adaptation according to environmental changes.

Our first contribution in this paper is the adoption of the Adaptation Space concept [9] to manage the proactive generation of query plans. These parameterized plans will service as alternatives to react to unexpected shortages of runtime resources. In addition to Adaptation Space, we also use a systematic method based on feedback to monitor the runtime environmental variables for significant changes. Using Adaptation Space and feedback, we develop an integrated adaptation methodology combining proactive and reactive adaptive query processing, including policies and mechanisms for determining when to adapt, what to adapt, and how to adapt.

The second contribution of this paper is an experimental evaluation of query adaptation trade-offs in the presence of *network delays*. We show that by intelligent switching to new query plans, we may increase concurrency to bypass bottlenecks caused by network delays and latency fluctuations. As a result, query response time can be improved up to 40%. Our study evaluates both the gains as well as the limitations of query adaptation. We observe that the net gains of query adaptation are primarily due to the added concurrent processing. However, adaptation gains are often bounded by other runtime resource constraints, such as available memory. Our experimental results show that *Ginga* is a promising approach for implementing query adaptation in open environments such as the Internet.

The rest of this paper is organized as follows. Section 2 motivates the *Ginga* approach using a real world example. Section 3 presents an overview of *Ginga* system. Section 4 describes the experimental setup and Section 5 presents the experimental evaluation results. We discuss related work in Section 6 and conclude the paper in Section 7.

2. MOTIVATING EXAMPLE

Before describing the *Ginga* approach to query adaptation, we discuss our experiences and observations obtained from

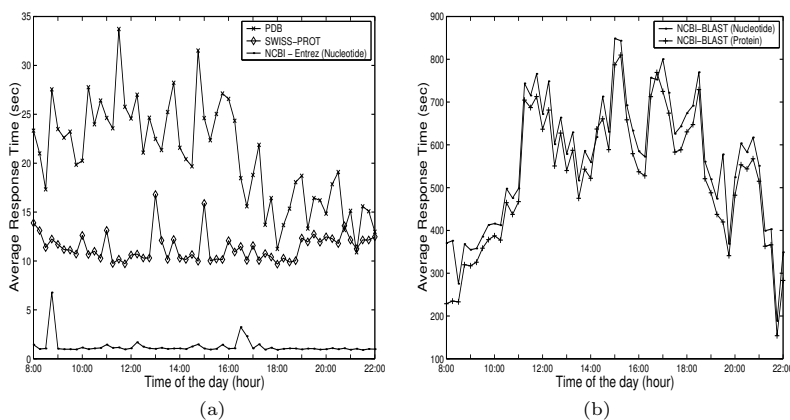


Figure 1: Average response times measured every 15 minutes for 32 days (from 4/11/2002 to 5/13/2002) between 8:00 and 22:00 EST. (a) Keyword query on “HIV” using PDB, SWISS-PROT, and NCBI Entrez; (b) NCBI BLAST searches on nucleotide or protein sequences, matching a nucleotide or a protein sequence found in HIV.

our participation in a DoE SciDAC¹ multi-discipline research initiative.

It is well known that there are more than 500 Bioinformatic data sources accessible on the Internet. One of the common applications of this growing number of Bioinformatic data sources is the discovery of new drugs that may efficiently treat serious diseases caused by virus such as Human Immunodeficiency Virus (HIV). As we describe next, such applications will involve contacting different web services that have unpredictable response times. The main challenge in this scenario is to execute a user request over these unstable remote services with expected responsiveness. **Example 1 (Concrete Application):** Consider Next Generation Drugs (NGD), a company specialized in developing state-of-the-art drugs. For each new drug that NGD starts to develop, the first step is to gather all the needed information about the virus that the new drug will attack. In general, this information describes the nucleotides, proteins, and protein structures associated with the virus along with assay results on how chemical compounds can affect the virus proteins. Further information about the virus can be obtained by performing BLAST (Basic Alignment Search Tool) searches, which finds similar protein sequences to those found in the virus. This information may be relevant, for example, while determining what would be the possible side effects of the new drug over other similar proteins.

Typically, collecting the information described above will require contacting remote web services such as NCBI Entrez² for data on nucleotides, SWISS-PROT³ for proteins, PDB⁴ for protein structures, and NCBI BLAST⁵ for similar protein sequences. However, the response time from these web services can be very unpredictable as shown in Figure 1. Even though NCBI Entrez has fast response time (in the order of a few seconds), waiting for PDB results can take up

to 34 seconds depending on what time of the day the query request was posted to the site. Therefore, if the data collected from different web services have to be combined or integrated before delivering to the users, such difference in response time could become costly. The situation may become aggravated when NGD needs to execute this type of queries repeatedly (practically, one for each new chemical compound that can potentially affect the virus). The accumulative effect of these delays can significantly affect the company’s productivity.

One possible solution that NGD may use is the Ginga system, which uses the two-phase distributed query adaptation mechanisms in the presence of network delays and latency fluctuations. Our experimental analysis shows that the Ginga approach to query adaptation is efficient and effective.

3. SYSTEM OVERVIEW

*Ginga*⁶ query adaptation engine [13] is a distributed software system that supports adaptive query processing. Like the rhythm and movements of samba, with *Ginga* adaptation engine we want to efficiently change the execution of a query plan to keep up with the rhythm imposed by runtime variations in the environment.

Ginga system architecture is depicted in Figure 2. Every query submitted to *Ginga* is initially processed by the *query manager*, which prepares the *Ginga* system for the performance optimization process. One of the important tasks of the query manager is query routing [10] of each user request. Query routing is dedicated to prune those data sources that cannot directly contribute to the answer of the query. After query routing, each end-user query is transformed into a set of subqueries associated with some execution dependencies. Each subquery is targeted to one of the chosen data sources.

Ginga query adaptation engine combines a *proactive* engagement phase, before query execution, with a *reactive* control phase during the execution. Before the query starts,

¹<http://www.science.doe.gov/scidac/>

²<http://www.ncbi.nlm.nih.gov/Entrez>

³<http://ca.expasy.org/sprot>

⁴<http://www.rcsb.org/pdb>

⁵<http://www.ncbi.nlm.nih.gov/BLAST/>

⁶*Ginga* is a Brazilian word typically used to describe a quality that a person needs to have when dancing *samba*, the famous Brazilian rhythm.

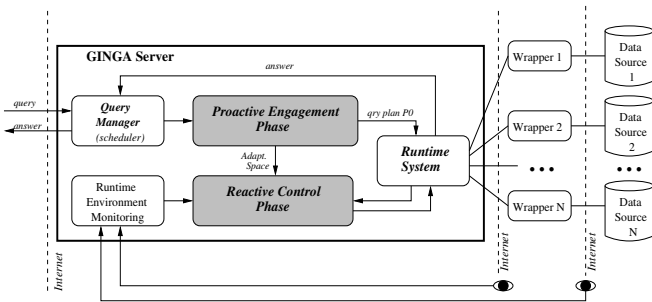


Figure 2: GINGA System Architecture.

Ginga builds an initial optimized plan using the resulting queries from query routing and generates some alternative execution plans that may be needed due to runtime variations in the environment. During query execution, Ginga monitors the system resource availability through execution progress, determines *when* to change the query plan and *how* to adapt by choosing an alternative plan created in the proactive phase.

Proactive Engagement Phase

Proactive engagement phase consists of two main steps. First, Ginga builds an initial optimized query plan P_0 using any existing query optimization algorithm for distributed databases (e.g., [17, 15, 11]). Second, Ginga establishes a selection of alternative query plans ($\{P_i, i = 1, \dots, n\}$) for adaptation. Since there are many potential alternative plans, we organize them into an adaptation space.

Adaptation Space Model

In this section, we informally present the key components of the adaptation space model as applied to query adaptation and illustrate the construction process of an adaptation space that coordinates query adaptation in the presence of network delays. Formal definitions for the adaptation space model are presented in [9].

In general, an adaptation space consists of two main components: the set of adaptation cases and the set of adaptation triggers. In this paper, adaptation cases are the query plans $\{P_i, i = 0, \dots, n\}$, each optimized for a set of environmental parameters. An adaptation trigger is defined as a quadruple $\langle P_{from}, AT_condition, P_{to}, wait_time \rangle$. Assuming that Ginga is currently executing the query plan P_{from} , when $AT_condition$ (a significant runtime environment change such as network delay) happens, Ginga adapts by making the transition from P_{from} to P_{to} . The $wait_time$ component indicates for how long $AT_condition$ must hold before the described transition takes place.

An adaptation space can also be viewed as a *transition graph* – a *lattice* with nodes representing query plans and edges representing the transition from one query plan to another. Below, we illustrate the construction of an adaptation space by using the application scenario described in Section 2.

Example 2 (Adaptation Space Construction): Assume that NGD uses Ginga to add query adaptation into their query processing system. Now the queries from pharmacologists can be processed more efficiently in the presence of network delays. Suppose that a pharmacologist wants to investigate new drugs to treat HIV. As a first step, she needs to issue a query Q to gather the following information about the HIV virus: nucleotides, proteins, and protein

structures associated with the virus along with the related assay results. For illustration purposes, we assume that Q is expressed in a SQL statement as shown in Figure 3(a). However, any other query language, such as XQuery, could very well be used instead.

In order to process Q , Ginga first identifies the bioinformatic sources with the information needed (query routing) and then generates the initial optimized query plan P_0 (Figure 3(a)) to process Q . For simplicity, we assume that only the following sources were selected to answer Q : *NCBI* for nucleotides, *SWISS-PROT* for proteins, *PDB* for protein structures, and *RemoteLab* for the related assay results⁷. Also, assume that the expected transfer rate for the network connection L_i is $Rate(L_i) = 1Mbps$, for $1 \leq i \leq 4$. Ginga will adapt the execution of Q whenever $Rate(L_i)$ drops below $w \times 1Mbps$, $0 < w \leq 1$.

The construction of an adaptation space for executing Q has three main steps. First, we generate P_0 and record the assumptions made about the runtime environment used for optimizing P_0 . Second, we create the *important* network delay scenarios ($AT_conditions$), when adaptation is needed to recover from the consequent execution delay. Third, we construct the transition graph by generating the needed query plans for each $AT_condition$. The result is shown in Figure 3(b). P_0 is at the top (labeled as $ac_initialP$) and the other nodes represent alternative query plans when network delays occur.

It is important to note that it is unrealistic to construct an adaptation space covering all possible adaptation cases. A more realistic approach is to generate adaptation cases for only those runtime environment changes that are known to occur with a frequency above a realistic threshold.

Reactive Control Phase

The reactive control phase takes as input the associated adaptation space for the query Q being executed and initiates the monitoring of $AT_condition$ specified in Q 's adaptation triggers that are relevant to the current query plan P_{from} (e.g., transfer rate of network connections). The adaptation process is triggered when at least one $AT_condition$ becomes true (e.g., network delay). We call the transition from plan P_{from} to P_{to} an *adaptation action*.

In general, there are two kinds of $AT_conditions$: *content-based event push* and *time-based event push*. With content-based event push, the evaluation of $AT_condition$ periodically observes and records the environmental parameters specified in the $AT_condition$. If an $AT_condition$ becomes true for more than $wait_time$, the adaptation process to switch the query plan to P_{to} starts. With time-based event push, Ginga (reactive control phase) sets a timeout for each remote data sources (e.g., long initial connection cost). If timeout happens, a network delay is detected directly. In the current prototype of Ginga, we use the $wait_time$ as the timeout value for detecting network delays. In case multiple $AT_conditions$ become true, the time-based event push has higher priority due to its immediate validity.

We now describe the concrete adaptation actions taken by Ginga to cope with network delays caused by slow delivery, a possible runtime environment change. Upon detecting slow delivery, Ginga first reacts by scheduling adaptation actions that involve materializing independent subtrees

⁷In this example, we assume that NGD has many different labs geographically distributed, where each lab has its own database of assay results.

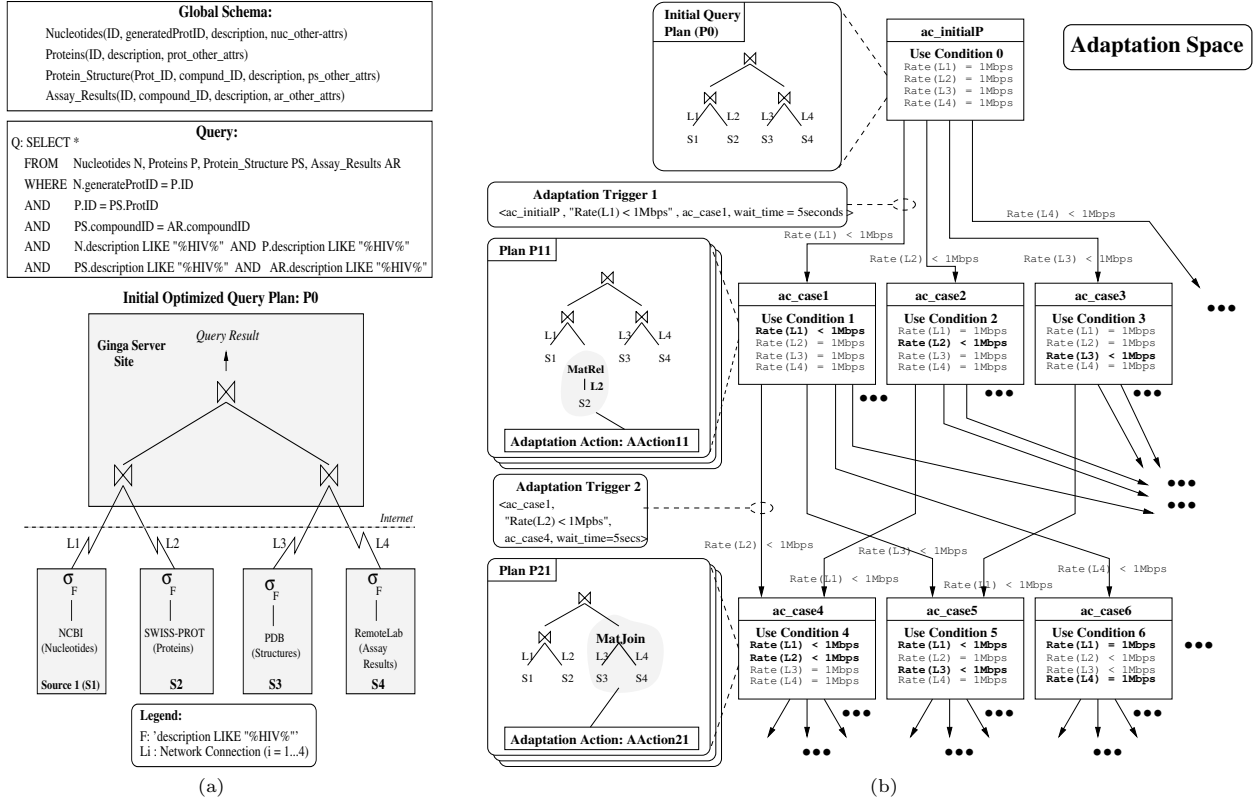


Figure 3: (a) Query Q for collecting information on HIV and (b) associated adaptation space for Q .

from the original query tree while continuing to process the data retrieval through the slow network connection⁸. When no more independent subtrees can be materialized and the problematic connection is still slow, Ginga schedules adaptation actions that create and materialize new joins between the relations that were previously materialized.

Example 3 (Adaptation Action): Suppose that as Ginga initiates the execution of query plan from Figure 3(a), slow delivery is detected on connection L_1 . In order to cope with this network delay, Ginga schedules the first adaptation action to start plan P_{11} , which concurrently materializes the search results from *SWISS-PROT* ($MatRel(SWISS-PROT)$). If Ginga finishes processing *NCBI* search results while concurrently running $MatRel(SWISS-PROT)$, then no further adaptation is necessary. Otherwise, another adaptation action should be scheduled. In this example, the new query plan P_{12} (Figure 4) starts concurrently the materialization of join between the search results from *PDB* and *RemoteLab* ($MatJoin(PDB, RemoteLab)$)⁹.

When there are no more independent query subtrees from the original plan to be materialized and Ginga still has not finished processing *NCBI* results, the next adaptation action

is to start with joins. In this example, the new query plan P_{13} will create and materialize a new join with the relation and join results that P_{12} materialized. In order to avoid Cartesian products, Ginga creates new joins by following the query graph associated with P_0 .

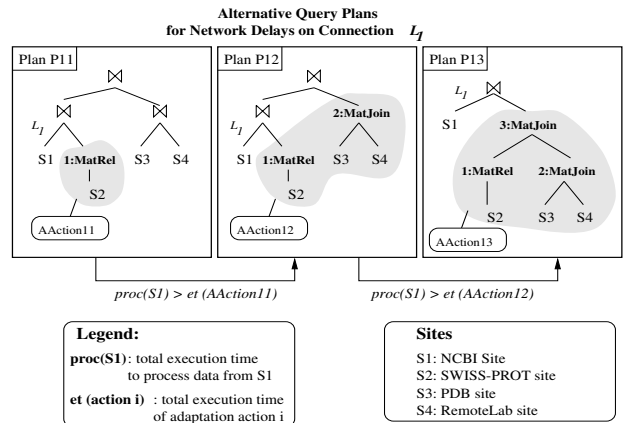


Figure 4: Alternative query plans for coping with network delays on connection L_1 .

⁸We call *independent subtree* a query subtree from the original plan that does not depend on the input from the delayed source.

⁹The materialization operations in each plan are numbered in the order they should be executed.

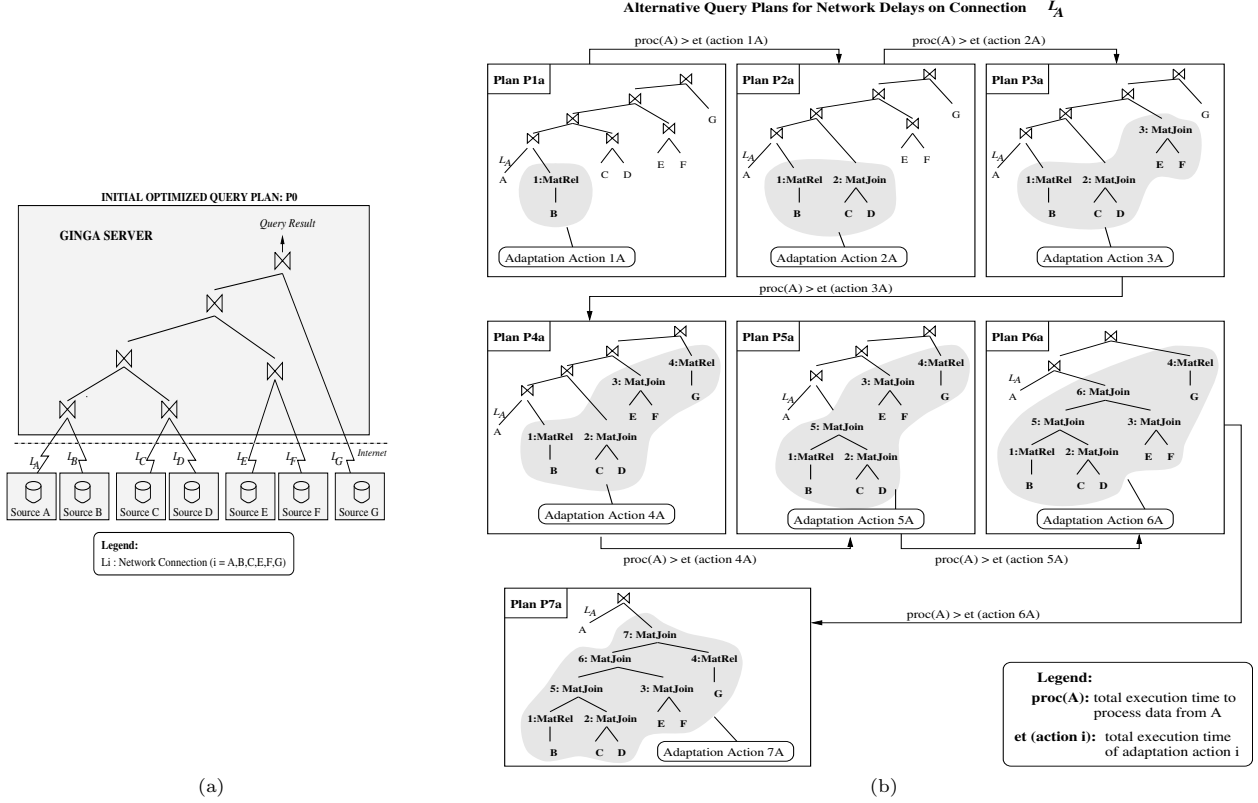


Figure 5: (a) Initial optimized plan P_0 used for the experiments on network delays.; (b) Adaptation actions for coping with delays on network connection L_A .

4. EXPERIMENTAL SETUP

For all experiments reported in this paper, we use a simulator (based on the CSIM toolkit) that models a client-server system with one client running Ginga and seven remote data servers. We assume that the servers are not handling any workload other than responding to Ginga requests.

Table 1 lists the main parameters used for configuring the simulator. Client and server machines have the same hardware configuration. Disks are modeled as FIFO queues. We model the network as point-to-point communication links between the client and each data server. Network connections are independent of each other, in the sense that the failure of a connection does not affect the others. The normal bandwidth of each connection is $5Mbps$ and degrades to $500Kbps$, $256Kbps$, and $128Kbps$.

For our experiments, we use the initial query plan P_0 depicted in Figure 5(a), where each remote source is expected to return 10,000 objects of 200 bytes each. We use a bushy tree because it allows us to fully investigate all aspects of Ginga adaptation engine. Nevertheless, similar results were obtained for left-deep and right-deep query trees.

We assume that relations can be equijoin in any order, always on the same attribute. We consider two sizes of memory for executing the query at the Ginga site: *limited* and *unlimited*. With *limited* memory, all operators are executed with the minimum memory requirement, while with *unlim-*

ited memory the operations are executed at their optimal performance. For example, in a hash-join between relations R and S the minimum and maximum memory requirements are respectively $\sqrt[3]{|R| \times F}$ and $|R| \times F$, where $|R| < |S|$ and F represents the overhead factor due to the hash structure [16].

Table 1: Simulation Parameters

Parameter	Value	Description
Speed	100	CPU speed (MIPS)
PageSize	8192	disk page size (bytes)
SeqIO	3.5	per disk page for sequential I/O (msecs)
RandIO	11.8	per disk page for random I/O (msecs)
DiskIO	5000	instructions to start a disk I/O
NBANDW	5	network bandwidth (Mbps)
Move	2	instructions to move 4 bytes
Comp	4	instructions to compare keys
Hash	25	instructions to hash a key

5. EXPERIMENTAL RESULTS

In this section, we report the results from our study on the performance characteristics of Ginga’s reactive control phase. We first investigate the effectiveness of Ginga while coping with network delays. Then, we introduce errors to the selectivity of the newly created joins to explore the performance boundaries of Ginga System.

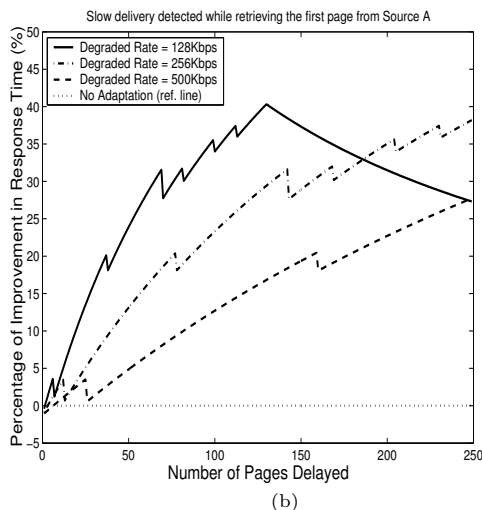
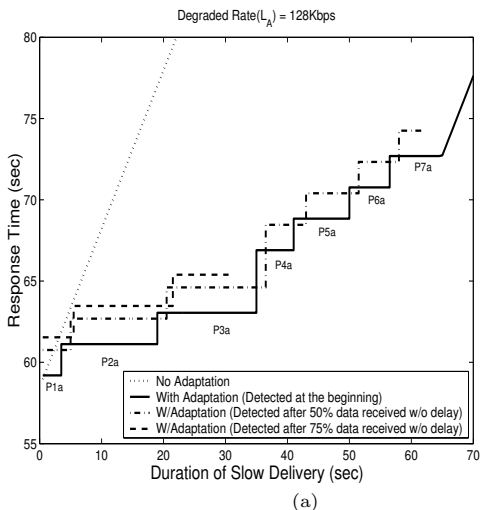


Figure 6: (a) Limited Memory ; Normal Rate = 5Mbps ; Degraded $Rate(L_A) = 128Kbps$. (b) Percentage of improvement in response time; Limited Memory; Degraded connection: L_A ; Slow delivery detected while retrieving the *first* page from remote Source A.

Effectiveness Analysis of Ginga Adaptation to Network Delays

The graph in Figure 6(a) shows the response time of Ginga’s adapted plans when slow delivery is *first* detected in three different situations while retrieving data from Source A: (1) at the beginning, (2) after 50%, and (3) after 75% of pages were received without delay. Once the delay is detected, we assume that the data is slowly delivered under a degraded rate $Rate(L_A) = 128Kbps$. The duration of the slow delivery is represented by the x-axis. The response time of the original plan (without adaptation) is represented by the dotted line. For the experimental results reported in this section, we assume that all intermediate results have 5,000 objects and the query is executed under *limited* memory size.

As we can see from the graph from Figure 6(a), the benefits of adapting the query execution in the presence of slow delivery are significant. This fact is illustrated by the “staircase” lines for the case when the adaptation process is used. The horizontal length of each step in each adaptation line represents the amount of slow delivery that each alternative query plan can absorb after Ginga schedules it. The height of each step shows the response time of the adapted query execution if $Rate(L_A)$ resumes to its expected data transfer rate of 5Mbps after a period of observed slow delivery. The label in each step indicates the query plan being used. A description of all alternative plans are depicted in Figure 5(b). For short duration of slow delivery (less than 4 seconds), Ginga can practically maintain the same query response time as the initial plan when executed without delay. This can be observed from the solid line in Figure 6(a). However, this scenario does not occur for the ‘50%’- and ‘75%’-lines. In fact, in these two cases, for slow delivery with very short duration the response time with adaptation is worse than the response time with no adaptation. We are currently investigating mechanisms to avoid this anomaly.

For slow delivery duration longer than 4 seconds, Ginga’s query adaptation is beneficial even when network delays are detected after 50% and 75% of pages were downloaded without delay. The longer the duration of slow delivery, the more processing can be done concurrent while downloading the slow data. However, we observe that there are a few moments when using adaptation can result in a query response time that is *almost* as bad as no adaptation. One of these moments can be observed at 5 seconds with the ‘50%’-line.

The improvements provided by Ginga adaptation process is limited by the number of alternative query plans. When it is no longer possible to fully process the slow data while executing the sequence of materialization operations from query plan P_{7a} , there is nothing left to be done other than process the slow data as it arrives. This explains why after 65 seconds of slow delivery the solid-adaptation line becomes parallel to the no adaptation line. However, Ginga adaptation is still beneficial even though it can no longer absorb all the slow delivery.

The number of needed alternative query plans is reduced according to when the slow delivery is first detected and how seriously the $Rate(L_A)$ is degraded. From the ‘75%’-line in the graph of Figure 6(a), we can clearly see that Ginga adaptation process uses only up to query plan P_{3a} . This occurs because when the network delay is detected there are only a few pages left from A that need to be processed concurrently with the sequence of scheduled materializations. In other words, less critically the $Rate(L_A)$ is degraded and later the slow delivery is detected, fewer the alternative query plans that are needed.

Figure 6(b) shows the comparative benefits of using Ginga versus no query adaptation when slow delivery is first detected at the beginning under three different degraded rates for connection L_A : 128Kbps, 256Kbps, and 500Kbps. The x-axis represents the number of pages delayed, and the y-axis represents percentage of improvement in response time obtained by using Ginga. As we can see from the graph, in

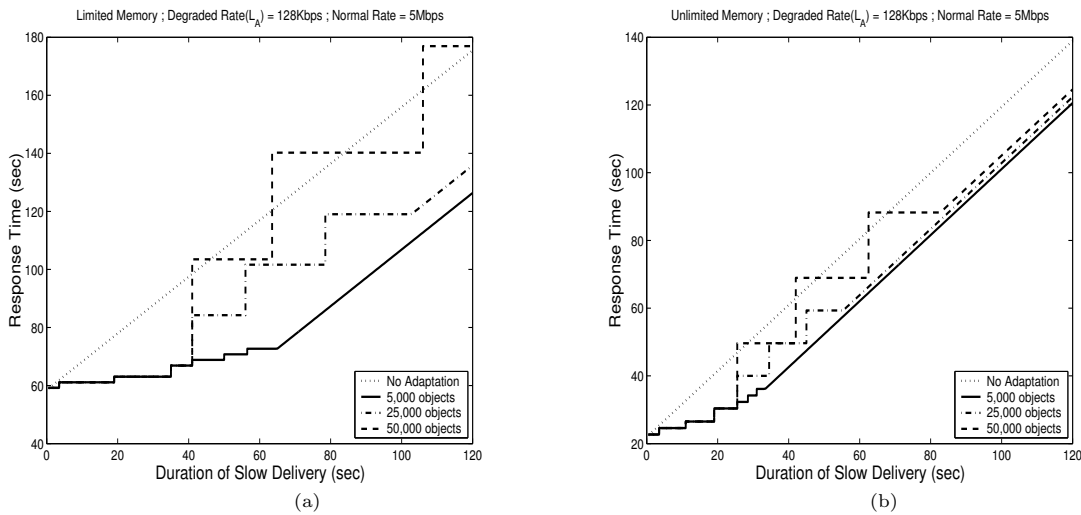


Figure 7: Ginga’s performance under different result sizes for the newly created joins. (a) *Limited Memory* ; Normal Rate = 5Mbps ; Degraded $Rate(L_A) = 128Kbps$; (b) *Unlimited Memory* ; Normal Rate = 5Mbps ; Degraded $Rate(L_A) = 128Kbps$.

this experiment Ginga can provide up to 40% of improvement on the query response time when compared with the no adaptation case.

The spikes in each curve in Figure 6(b) represent the moments at which Ginga switches plan. When there are no more alternative plans to schedule and the number of pages delayed increases, the response time improvement due to adaptation starts to gradually decrease. This occurs because the last scheduled plan can no longer fully absorb the processing of the slow data. Line ‘128Kbps’ represents this scenario. In contrast, for the ‘256Kbps’ and ‘500Kbps’ lines, Ginga never experiences a situation where the alternative query plans cannot fully absorb the slow delivery.

The results from the experiments with slow delivery detected for the other network connections of query plan P_0 (Figure 5(a)) are similar to the results described so far in this subsection. Due to space limitations, we omit these results.

Ginga Performance Boundaries

We now analyze Ginga’s performance boundaries when significant variance is observed for the selectivity of the newly created joins used by some of the alternative query plans. We consider two situations: when the memory is *limited* and when memory is *unlimited*. With limited memory, some of the query plans require a large number of disk I/Os which can lead Ginga to worse performance than no adaptation. On the other hand, with unlimited memory, I/O costs are not incurred, but CPU power processing can be a bottleneck for Ginga’s adaptation process.

Figure 7(a) shows the experimental results under *limited memory* when newly created joins used on query plans P_{4a} , P_{5a} , and P_{6a} (see Figure 5(b)) produce 5,000; 25,000; and 50,000 objects. The ‘5,000’-line in the graph represents the situation studied previously. For new joins resulting in 25,000 objects, the response time with adaptation is still acceptable. However, when compared with the ‘5,000’-line, the response time for 25,000 objects increases significantly

(up to 32%) because for each new join, it is necessary to partition the relations due to the limited memory size. This situation is aggravated when the new joins produce 50,000 objects. In this case, there are moments when the response time with adaptation becomes worse than the response time with no adaptation. In other words, this means that waiting for $Rate(L_A)$ to resume to its expected rate is better than adaptation. Similar results were obtained for different degraded rates and having slow delivery first detected at different moments.

Figure 7(b) shows the experimental results under *unlimited memory*, which are analogous to the results in Figure 7(a). By assuming unlimited memory size, all the joins are executed in main memory eliminating I/O costs. However, note that unlimited memory still does not prevent the ‘5,000’-line from crossing the no-adaptation line. Intuitively, from this result, we conclude that Ginga’s adaptation process is not only memory bounded, but also CPU bounded. Adapting the query execution to memory and CPU constraints is part of our ongoing research study.

6. RELATED WORK

Adapting the execution of queries due to changes to the runtime environment parameters has been an important area of research, starting from early 90’s [6, 5], and continuing to today [2, 1, 7, 8, 3, 4]. Depending on *when* the adaptation takes place, previous approaches to query adaptation can be broadly categorized into *proactive* and *reactive* methods [1]. Proactive methods adapt query execution at start-up time (i.e., while loading the query plan to be executed) based on the current runtime environment, while reactive methods adapt the query execution at runtime by reacting to runtime environment changes. In contrast, Ginga combines proactive and reactive adaptation.

Two classical proactive adaptive methods are Volcano Dynamic Plan [5] and Parametric Optimization [6]. Volcano Dynamic Plan uses *choose-plan* operators to enable the op-

timizer to cope with the inability to precisely estimate all the resource parameter values at compile-time. Parametric Optimization attempts to generate one optimized query execution plan for each possible combination of values for the resource parameters that are unknown at compile-time. In comparison, Ginga can use these methods in the static query plan generation phase. In addition, Ginga has run-time reactive adaptation.

Two reactive methods closely related to Ginga are Query Scrambling [1, 2] and Dynamic Scheduling Execution (DSE) [4]. Query Scrambling uses materialization and operator synthesis to adapt the execution of distributed queries in the presence of network delays. The reactive method implemented by the DSE also adapts the execution of distributed queries in the presence of network slow delivery. In comparison, Ginga uses a unified framework (adaptation space model) to support both proactive and reactive adaptation. Ginga's framework is powerful enough for us to study not only query adaptation to network delays, but also query adaptation to memory and CPU constraints.

There are several other approaches to reactive adaptive methods. Mid-query re-optimization [8] attempts to re-optimize the query execution whenever a significant difference between estimated and observed values for the resource parameters is detected at runtime. The Tukwila project [7] proposes query (re-)optimization based on rules defined over possible runtime environment changes. Eddies [3] suggests the re-ordering of operators in the presence of configuration fluctuations of the runtime environment during query execution. While these projects propose more sophisticated mechanisms to support adaptation, Ginga uses a unified simple model and feedback-based monitoring mechanism to support both proactive and reactive adaptation using the existing relational operators.

7. CONCLUSION

Adaptive query processing is an active research area that has received considerable attention (see Section 6) due to the increasing openness of information systems such as Internet, where sudden changes such as network delays are common. Most of the previously proposed approaches can be classified into proactive (compile-time or start-up time generation of foreseeable query plans) or reactive (runtime adaptation according to actually observed environmental changes). In this paper, we described Ginga, an adaptive query processing engine that combines proactive generation of query plans with run-time monitoring and reactive switching of query plans. We evaluate the effectiveness of Ginga primarily under network delays, with particular attention to slow delivery, a common problem that has received little attention previously.

At compile-time, Ginga uses the Adaptation Space concept to organize alternative query plans, generated for the main cases of network delays. At runtime, Ginga relies on feedback mechanisms to monitor the query execution to detect environmental changes. Our experimental evaluation shows that Ginga achieves almost always significant performance improvements compared to no adaptation.

Our results show that Ginga holds promise both as an approach and as a system for adaptive query processing. Ginga is a good approach since Adaptation Space supports a variety of adaptation actions through a uniform framework. Ginga is a good system due to the effectiveness of

its adaptation policies and mechanisms, as demonstrated by our experimental evaluation. We are currently extending the Ginga work to study other kinds of system resource constraints (e.g., CPU and memory) and their interactions with network delays.

8. ACKNOWLEDGEMENTS

The first author was partially supported by CAPES - Brasilia, Brazil - and DoE SciDAC grant. The other two authors were partially supported by DARPA, DoE, and NSF grants.

9. REFERENCES

- [1] L. Amsaleg, M. J. Franklin, and A. Tomasic. Dynamic query operator scheduling for wide-area remote access. *Journal of Distr. and Parallel Databases*, 6(3), 1998.
- [2] L. Amsaleg, M. J. Franklin, A. Tomasic, and T. Urhan. Scrambling query plans to cope with unexpected delays. In *PDIS*, 1996.
- [3] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *ACM SIGMOD*, 2000.
- [4] L. Bouganim, F. Fabret, C. Mohan, and P. Valduriez. Dynamic query scheduling in data integration systems. In *ICDE*, 2000.
- [5] R. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In *Proceedings of the ACM SIGMOD'94*.
- [6] Y. E. Ioannidis, R. T. Ng, K. Shim, and T. K. Sellis. Parametric query optimization. In *VLDB*, 1992.
- [7] Z. G. Ives, D. Florescu, M. Friedman, A. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *ACM SIGMOD*, 1999.
- [8] N. Kabra and D. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *ACM SIGMOD*, 1998.
- [9] L. Liu. Adaptation cases and adaptation spaces: Notation, issues, and applications (part i: Concepts and semantics). Technical report, OGI CSE Heterodyne Project, 1998.
- [10] L. Liu. Query routing in large-scale digital library systems. In *ICDE*, 1999.
- [11] L. Liu, C. Pu, and K. Richine. Distributed query scheduling service: An architecture and its implementation. *JCIS*, 7(2-3), 1998.
- [12] L. Liu, C. Pu, and W. Tang. Continual queries for internet-scale event-driven information delivery. *IEEE Knowledge and Data Engineering*, 1999. Special Issue on Web Technology.
- [13] H. Paques, L. Liu, and C. Pu. Ginga: A Self-Adaptive Query Processing System. (short paper) In *CIKM'02*.
- [14] D. A. Schneider and D. J. DeWitt. Tradeoffs in processing complex join queries via hashing in multiprocessor databases machines. In *VLDB*, 1990.
- [15] P. G. Selinger and M. Adiba. Access path selection in distributed database management systems. In *VLDB*, 1980.
- [16] L. D. Shapiro. Join processing in database systems with large main memory. *ACM TODS*, 11(3), 1986.
- [17] M. Stonebraker. The design and implementation of distributed ingres. *The INGRES Papers*, M. Stonebraker (ed.)(Addison-Wesley), 1986.