Towards the Study of Performance Trade-offs Between Materialized and Virtual Integrated Views *

Richard Hull
Computer Science Department
University of Colorado
Boulder, CO 80309-0430
email: hull@cs.colorado.edu

Gang Zhou†
Computer Science Department
University of Colorado
Boulder, CO 80309-0430
email: gzhou@cs.colorado.edu

Abstract

Consider the problem of supporting an integrated view over multiple databases. The traditional approach is to use a virtual view, but recent investigations are proposing to use a materialized view, or a hybrid virtual/materialized view. This paper initiates an investigation into the performance trade-offs along this spectrum of choices. In particular, the paper develops analytical models for predicting query-response time, view freshness, and system load in terms of parameters such as query frequency, query complexity, update frequency, and network delay. In many ways the performance of a mediator-based integration environment is similar to that of a client-server DBMS architecture. However, the notion of query freshness does not explicitly arise in a single DBMS, and complex joins may more likely in an integration environment.

This paper lays the groundwork for conducting benchmarking experiments for integrated views, and presents the results of some initial experimentation. Such results will be used to calibrate the analytical model for specific application environments. This will enable prediction of the behavior of the virtual and materialized approaches in specific contexts.

1 Introduction

The advent of the Information Superhighway has dramatically increased the need for efficient and flexible mechanisms to provide integrated views over multiple information sources. The traditional approach to this problem is to represent the view in a *virtual* fashion; queries against the view are decomposed and sent to the remote sources (e.g., [T+90, ACHK93]). More recently, a complementary approach has emerged, that is based on storing the view in *materialized* form [WHW89, ZHK96, ZGHW95]. In that approach, queries can be answered without accessing the source databases, and the materialized view is typically maintained via propagation of incremental updates. Reference [HZ96] combines the virtual and materialized paradigms, by describing how a broad variety of hybrids of them can be used to support integrated views.

This paper initiates an investigation of the the relative advantages and trade-offs of the different points along the spectrum of choices between fully virtual and fully materialized. The approaches are compared with respect to query-response time, view freshness, and system load, in terms of parameters such as query frequency, query complexity, update frequency, and network delay. An important contribution of this study is the identification of the key performance characteristics that should be considered when choosing between the materialized and virtual approaches. Another contribution is the development of formulas describing the behavior of simple integrated views as various system parameters are varied. These formulas will permit focused benchmarking experiments, the results of which can be used to calibrate the formulas to specific environments. The results of some initial benchmarking experiments are presented here.

There are many similarities between the use of mediators to support integrated views across multiple databases, and the use of a client-server architecture within a single DBMS environment (possibly with multiple servers). In particular, the trade-offs in the client-server architecture associated with query-shipping vs. data-shipping and caching (e.g., [DR93]) closely parallel the trade-offs in the mediator architecture between the virtual and materialized approaches. Furthermore, the use of materialization in supporting views in a conventional DBMS architecture (e.g., [Han87]) has similarities to the use of materialization in mediators. A key difference between the mediator context and the other contexts is that with mediators, the source databases are presumed to be largely autonomous from each other and from the mediator. In particular, mediated environments typically do not support global transactions (see [HZ96] for a discussion of view consistency in this context). As a result, view freshness becomes an important performance characteristic distinct from response time. A minor difference between the mediated and other environments is that in the former, the various systems are likely to be connected by a wide-area network. Thus

*This research was supported in part by NSF grant IRI-931832, and ARPA grants BAA-92-1092 and 33825-RT-AAS.

†A student at the University of Southern California, in residence at the University of Colorado.

network delay and variability can become more of a factor. Another, more subtle, difference concerns the type of joins that may arise in a mediator context. If two databases to be integrated were developed by different organizations, there may not be universal keys for some families of objects. (E.g., one database might use social security number to identify people, and another one use nationality and passport ID number). Forming the join across relations in this case raises the issue of "object matching", and must often rely on user-defined functions [ZHK96]. For this reason we consider here both equi-joins and more complex joins.

For the first phase of the investigation of performance trade-offs between materialized and virtual views in mediated environments, we focus on the relational model. Further, we focus on two fundamental and representative operators: selection (against a single source relation) and join (against two relations from different source databases). Selection is representative of projection, and join is representative of other binary operators (union, intersection and difference). Simple equi-joins and more complex kinds of joins are considered.

In connection with join, we focus on three fundamental and representative implementations: (a) virtual; (b) materialized with fully materialized support [ZHK96], where auxiliary data is stored with the integrated view, so that the data sources do not have to be queried when processing the incremental updates; and (c) materialized but with no materialized support (e.g., see [ZGHW95]). A view with fully materialized support has been termed "self-maintainable" in [GJM96]. Views based on selection are self-maintainable, and so cases (b) and (c) above collapse for selection. (This is also true of projection, if a bag semantics is used).

This paper presents formulas describing the impact of different system parameters on the performance of integrated views based on selection and join. Also presented are initial benchmarking experiments, that provide limited confirmation of the formulas. One direction of our ongoing research is to explore the natural generalization of the results described in this paper to views based on arbitrary relational algebra expressions.

Section 2 gives background and the experimental framework for the investigation. We then consider three fundamental performance characteristics of integrated views: query response time (Section 3), view freshness (Section 4), and system load (Section 5).

2 The Experimental Framework

This section briefly describes the experimental framework that will be used in the first phase investigation of performance trade-offs between using the virtual and materialized paradigms to support integrated views. Specifically, this section identifies two fundamental kinds of integrated views, and a small family of representative approaches for supporting them. The section outlines the key algorithms used by these approaches; these will be used when establishing the performance characteristics of the integrated views they support. The algorithms presented here are based on algorithms of [HZ96] that provide a uniform approach to supporting materialized, virtual, and hybrid views. The section concludes with a brief description of the envirnoment used for our initial benchmarking experiments.

Following [Wie92], we assume that a *mediator* is used to support an integrated view over multiple databases. A mediator is a software component that can support restructuring and merging of information. In our context, the mediators are built from DBMSs, possibly with special capabilities, such as activeness. We restrict attention here to mediators that are separate from the source databases, although this is not a strict requirement. Also, an *integration environment* is a distributed software system involving source databases, a mediator that supports an integrated view over them, and the network that supports communication.

As noted in the Introduction, we focus here on support for integrated views defined by selection and by join. With regards to selection, we consider virtual and materialized views. The algorithms used for supporting these are straightforward: In the virtual case, queries against the view are translated into queries against the source database. The answers received are transmitted directly to the user. In the materialized case, queries against the view can be answered entirely within the mediator. Updates against the source database are transmitted to the mediator, which in turn performs incremental update propagation to incorporate the updates. In the materialized case, it is assumed that the mediator maintains a queue of deltas from the source database(s). Incorporation of the deltas in the queue might be eager (i.e., performed when the update is received) or lazy (by which we generally mean, performed on a periodic basis).

We turn now to join. There are two fundamental reasons that the study of selection does not provide sufficient information about join. One concerns the significant range in time needed to compute a join. In our framework we focus on two cases: simple equi-join, for which indexes in the source databases might be used, and which can be computed in time $O(n \log n)$ in the worst case; and a join with complex join condition (e.g., involving a user-defined function), where indexes in the source databases are essentially useless, and time $O(n^2)$ is needed to compute the join. (As noted in the introduction, such complex joins arise in connection with object matching.)

The second difference between join and selection is

that with join, there are a variety of approaches for supporting join views, in both the virtual and materialized contexts. We consider here three possibilities: (a) virtual using a *polling-based* algorithm, where the mediator independently polls both source databases for relevant portions of the relations to be joined (as in, e.g., SIMS [ACHK93]); (b) supported materialized; and (c) unsupported materialized. We now describe each of these in more detail.

In the polling-based virtual case, a join $\sigma_f(R) \bowtie_h \sigma_g(S)$ is computed as follows. (In the algorithms below, f' is a conjunction of f with relevant parts of h, and similarly for g'.)

1. send query $\sigma_{f'}(R)$ to database R, to obtain answer A_1;
2. simultaneously send query $\sigma_{g'}(S)$ to database S, to obtain answer A_2;
3. combine A_1 and A_2 (essentially, $A_1 \bowtie_h A_2$) to produce the final answer.

Although not considered here, an alternative to polling-based is *semi-join based*: query the first source database for relevant tuples, and then use these to form a semi-join query against the second source database. This approach was original developed in connection with distributed databases [BG81], and is useful in an integration context if the join selectivity condition h produces a very small number of tuples.

Figure 1(b) and (c) show two different approaches to providing a materialized join view $T = (\sigma_f(R) \bowtie_h \sigma_g(S))$. (Part (a) of the figure shows how a selection view is supported in the materialized case.) We typically assume that R and S are relations from different source databases. Part (b) of the figure shows the (*fully*) *supported materialized* approach: auxiliary relations $R' = \sigma_{f'}(R)$ and $S' = \sigma_{g'}(S)$ are materialized in the mediator. Under this approach, an update from the R database can be propagated into T without reference to the S database (and similarly for updates from the S database). More specifically, suppose that an incremental update ΔR against R is specified against the R database. We view ΔR as a set of insert atoms and delete atoms (e.g., see [GHJ+93, GHJ96]). To propagate this into the integrated view the following steps may be taken (using the bag semantics):

1. compute $\Delta R'$, i.e., the net effect of ΔR on R';
2. compute ΔT as $\Delta R' \bowtie S'$;
3. apply $\Delta R'$ to R' and apply ΔT to T.

The analogous steps can be taken for updates against S. In general, the updates against R and S will be stored by the mediator in a queue, and a set of updates will be processed by the mediator in a single transaction (e.g., see [ZHK96]).

Figure 1(c) shows the *unsupported materialized* approach, where only T is materialized. Our update propagation algorithm is similar to an algorithim in [ZGHW95], and uses compensation. But unlike that algorithm, our algorithm will always terminate. This algorithm requires that updates are transmitted to the mediator in an eager fashion (although they do not have to be propagated into the view in an eager fashion). We also assume that messages from a source database are received in the order in which they were sent. To permit maximum efficiency, we assume that the source databases are capable of supporting semi-join queries. The first step of update propagation is:

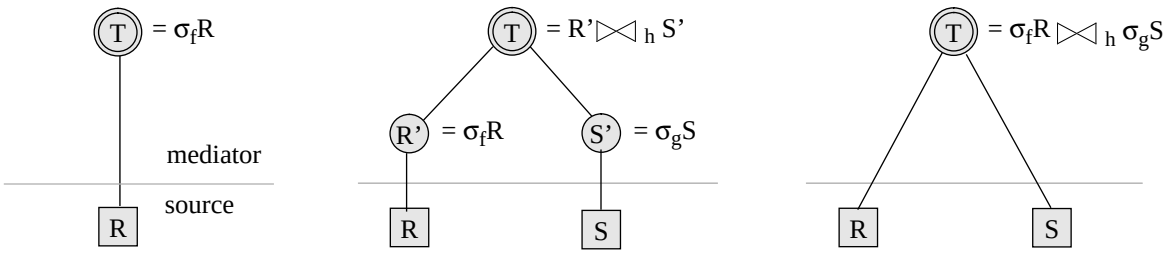
1. At time t_{prop}, the full set of updates from R and S in the queue is collected, to form ΔR and ΔS.

We now need to compute $\Delta T = (\sigma_f(\Delta R) \bowtie_h \sigma_g(S)) \ ! \ (\sigma_f(R) \bowtie_h \sigma_g(\Delta S)) \ ! \ \sigma_f(\Delta R) \bowtie_h \sigma_g(\Delta S)$, where binary operator '!' denotes *smash*, a specialized operator on deltas that corresponds to function composition (see [GHJ+93, GHJ96]). However, because of network and processing delay, it may be impossible to obtain the values of R and S corresponding to time t_{prop}. Processing proceeds as follows:

2. Poll the S database to obtain $\Delta T_1^R = \sigma_f(\Delta R) \bowtie_h \sigma_g(S)$. This will use the "current" value of S, which may reflect a time after t_{prop}. Compensate ΔT_1^R back to time t_{prop} by smashing with $\sigma_f(\Delta R) \bowtie_h \sigma_g([\Delta S']^{-1})$, where $\Delta S'$ is the net update on S received between times t_{prop} and the time when ΔT_1^R is received. (The operator '$^{-1}$' denotes taking the inverse of a delta.) Let ΔT_2^R be the result.
3. (Simultaneously) poll the R database and compensate, to obtain ΔT_2^S.
4. Compute ΔT according to the equation given above and using ΔT_2^R and ΔT_2^S, and apply it to T.

If the source databases cannot support semi-join queries, then in step 2 above, the full relation S would be retrieved, and similarly for step 3 above.

As noted earlier, we include in this paper the results of some initial benchmarking experiments. These experiments were performed on SUN SparcStations. Both mediator and source databases are implemented in the database programming language Heraclitus[Alg,C] [GHJ96], which stands on top of the Exodus DBMS toolkit [CDRS93]. Communication between the mediator and source databases was implemented using UNISYS KQML (e.g., see [FWW+]). The local area network was an ethernet used by approximately 30 machines, and the wide area network was the internet, with source databases in Boulder and mediator in Los Angeles.



(a) Selection (b) Supported join (c) Unsupported join

Figure 1: Support for materialized selection and join views

3 Response time

This section presents formulas describing how key system parameters affect the query response time of integrated views based on the materialized and virtual approaches. The section also proposes a family of benchmarking experiments that could be used to verify and calibrate the formulas, and presents some initial benchmarking results. Views defined by selection are considered in Subsection 3.1, and views defined by join are considered in Subsection 3.2. In both cases, we assume that the amount of incremental maintenance of materialized data (if any) is relatively small, so that the query response time is not materially affected. Although the analysis in this section is relatively straightforward, it provides an important foundation for the discussion of query freshness in Section 4.

3.1 Selection view

Let R be a relation in a source database, and let view T be defined as $T = \sigma_f R$. We focus on queries q against T having the form $q = \sigma_g T$.

There are three fundamental system characteristics contributing to the total query response time (the first and last apply only to the virtual case):

query transfer time (QTT) for transferring the query from the mediator to the source (only for the virtual approach).

query execution time (QET) for executing a query at either the source or the mediator, denoted by QET_{source} and $QET_{mediator}$, respectively.

answer transfer time (ATT) for transferring the answer of the query from the source database to the mediator (only for the virtual approach).

The response time (RT) for the virtual approach is given by:

$$RT_{select,virt} = QTT + QET_{source} + ATT,$$

while the response time for the materialized approach is equivalent to the QET, i.e.,

$$RT_{select,mat} = QET_{mediator}.$$

The response time of the materialized and virtual approaches is now considered for six different cases. The first case is called the *Base case*, and the other five cases are variations of it. The system characteristics for these cases are described shortly (see Table 1 for a summary). Benchmarking results of response times in milliseconds for the six cases are summarized in Table 3.1. These numbers were obtained by averaging the results of repeated executions of the same experiments. Although the specific time measurements varied due to changing system load (especially the network load), the averages give a general indication of the relative efficiency of different approaches supporting the views. A corresponding bar chart (not to exact scale) for the response times is presented in Figure 2. There are two bars for each case. The left bar represents the response time for the materialized views, while the right bar is for the virtual view.

We now discuss the response time for the six cases in turn. In the Base case, we assume $selectivity(f, R) = 10\%$ and $selectivity(g, T) = 1\%$ (i.e., the selection $\sigma_f R$ yields 10% of R, and similiarly for g and T); the mediator and the source are resident in the same Local Area Network (LAN); and q can take full advantage of indices in both the source and the mediator (in the materialized approach). For the virtual approach, the query q will be converted to $q = \sigma_{g \wedge f} R$ and shipped to the source. For the materialized approach, q will be evaluated at the mediator against the materialized relation T. The total response time in the materialized case is significantly shorter than that in the virtual case. First, there are no query transfer time and answer transfer time in the materialized case. Second, even the query execution time in the materialized case is shorter. This is because the materialized view T is only 10% of the size of the base relation R.

In the Big-Query case, the selectivity of the query

parameter	Base	Bigger-Query	Small-View	No-Index	M-Index	WAN
View selectivity	10%	10%	1%	10%	10%	10%
Query selectivity	1%	2%	1%	1%	1%	1%
Network	LAN	LAN	LAN	LAN	LAN	WAN
Source index	Yes	Yes	Yes	No	No	Yes
Mediator index	Yes	Yes	Yes	No	Yes	Yes

Table 1: Six cases for studying the response time of queries against the selection view

		Base	Big-Query	Small-View	No-Index	M-Index	WAN
virtual	QTT	98	98	98	98	98	405
	QET	612	1104	144	9743	9743	612
	ATT	1011	4794	131	1011	1011	5413
	total RT	**1721**	**5996**	**373**	**10852**	**10852**	**6430**
mat'ed	**total RT**	**301**	**874**	**193**	**1041**	**301**	**301**

Table 2: Breakdown of the total response time for selection views in milliseconds

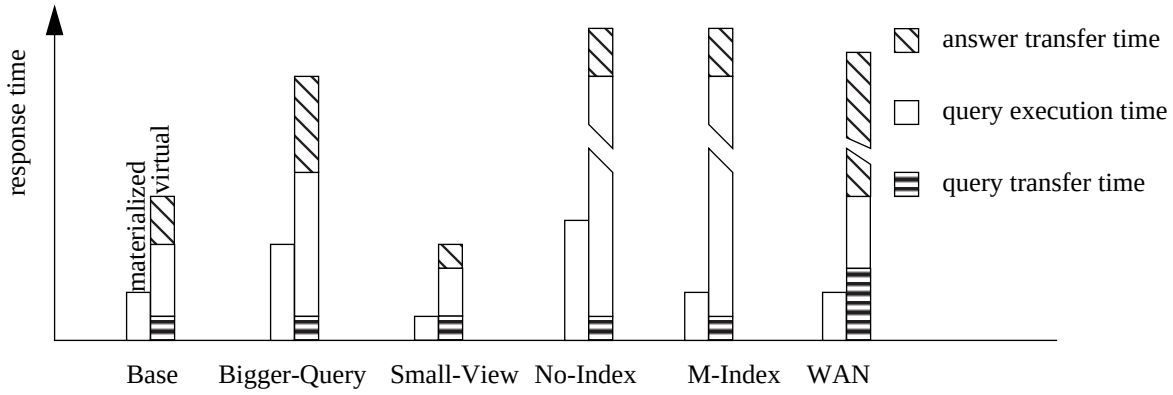


Figure 2: Response times of queries against the selection view in six different cases

q is set to 5% instead of 1% of the view. The main difference in response time is that the answer transfer time for the virtual approach is almost five times as long due to the increased selectivity of this case. The query execution time at both the mediator and the source do not increase nearly as much due to the use of indices.

The Small-View case differs from the Base case in terms that $selectivity(R, f)$ is set to 1% instead of 10%, i.e., the size of the view is much smaller than the one in the Base case. The benchmarking result shows that smaller view results in marginally shorter response time for the materialized view, and it results in shorter response time for the virtual view as well, because the size of result to be transferred from the source to the mediator is much smaller.

In the No-Index case both the relation R and the materialized view T in the mediator are not properly indexed for the query. The query is executed using sequential search, thus the QET is proportional to the number of the tuples of the relations. Since the tuple number of T is 10% of that of R, the QET for the materialized approach is also 10% of that for the virtual approach.

The mediator-Index case differs from the No-Index case in the way that only the materialized view T in the mediator is properly indexed. This is possible because of the lack of authority for the mediator administrator to create indices in the source. In this case, the QET for the materialized approach is much shorter than the corresponding QET in the No-Index case.

Finally, the WAN (Wide Area Network) case is a variation of the Base case, where the mediator and source are connected by a WAN instead of a LAN. Compared to the Base case, the query transfer time and answer transfer time longer longer due to the slower network. The query execution time for both the mediator and the source are not affected.

To summarize the above discussion, the materialized approach offers better response time in all six cases. This approach is especially favored when (1) the source is remotely located (WAN case), or (2) no adequate indices exist in the source to support queries against the view (No-Index and Mediator-Index cases).

3.2 Join views

In this subsection, we consider views defined by a binary join. The views have the form

$$T = \sigma_f R \bowtie_h \sigma_g S$$

where R and S are two relations from two autonomous source databases, f and g are select conditions, and h is a join condition.

Many characteristics of the response time of queries against the join view can be extrapolated from the study about the response time for selection views. We focus here on an issue that does not arise with selection views, namely significant differences in the amount of time needed to compute the join. We consider two kinds of join views, namely equi-join case and complex-join case. In both cases, we assume that $selectivity(f, R) = selectivity(g, S) = 10\%$, and the join condition h has the property that each of half of the tuples in $\sigma_f R$ matches exactly one distinct tuple in $\sigma_g S$.

We consider two selection queries against T. The first query has a 100% selectivity, and the second one has a 10% selectivity. Table 3 shows the benchmarking results of response times in milliseconds. Figure 3 shows a bar chart (not to exact scale) that compares the response time for the virtual and materialized approaches. The query against the materialized join view is simply a selection, which explains the very short response time. In a polling-based implementation described in Section 2, the total response time in the virtual case is:

$$RT_{join,virt} = QTT + QET_{source} + ATT + QET_{mediator}$$

Note that in the virtual case the summand $QET_{mediator}$, which is associated with the join of the results of the polling, is much larger than the total response time for the materialized approach. In the case of complex-join views, the benefit of a materialized view is even greater, because a complex-join can only be evaluated in $O(n^2)$ time, whereas an equi-join based on sort-merge algorithm can be evaluated in $O(n \log n)$ time in the worst case.

The 100% query selectivity experiment shows extreme difference in response time between virtual and materialized approaches. This is because in the virtual view case the mediator has to select a large amount of data from the sources and then do a join between two large relations on the fly.

4 Freshness of the view

A significant concern in connection with materialized integrated views is how "fresh" or "up-to-date" the view is. The complementary notion is "staleness". In this section we consider the issue of freshness and staleness of both virtual and materialized views. We begin by establishing formalism for describing staleness, and then establish formulas characterizing the worst-case staleness of various approaches to integration. We also provide preliminary benchmarking results.

4.1 Definition of staleness

For this discussion we assume that time is discrete. The state of a database DB at time t is denoted $state(DB, t)$. For a vector $\vec{DB} = (DB_1, \ldots, DB_n)$ of databases and vector $\vec{t} = (t_1, \ldots, t_n)$ of times, the state of $\vec{DB}$ at $\vec{t}$, denoted $state(\vec{DB}, \vec{t})$, is the vector $(state(DB_1, t_1), \ldots, state(DB_n, t_n))$. For this paper, we have $n = 1$ or $n = 2$.

		100% query selectivity		10% query selectivity	
		Equi-join	Complex-join	Equi-join	Complex-join
virtual	QTT	98	98	98	98
	QET_{source}	612	612	523	523
	ATT	1011	1011	212	212
	$QET_{mediator}$	19301	58234	1934	3041
	total RT	**21022**	**59955**	**2767**	**3874**
mat'ed	**total RT**	**824**	**824**	**431**	**431**

Table 3: Breakdown of the total response time for join views in milliseconds

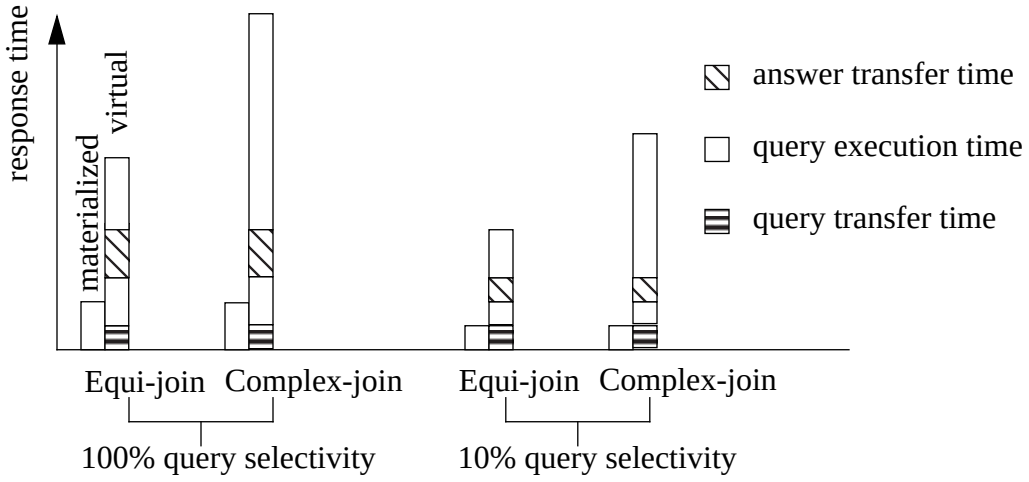


Figure 3: Response times of queries against join views

Suppose that V is a view over database vector $\vec{DB}$, with view definition ν. If global transactions are enforced, then the state of V at time t, $state(V,t)$, would essentially be $\nu(state(\vec{DB},\vec{t}))$, and the view state is always up-to-date. However, we consider here the context where the source databases and mediator are not coupled so tightly.

Suppose now that q is a query against view V. We blur notation somewhat, and also let q denote a specific *invocation* of query q. Two important times are t^q_{ask} when the query is asked and t^q_{get} when the answer is received. Also important is the time vector $\vec{t}_{der}$ holding the times of the source database states that the query answer was derived from. The formal definition of $\vec{t}^q_{der}$ is different depending on whether the view is virtual or materialized. Specifically, for database DB_i we have:

virtual: $t^q_{i,der}$ is the time that the answer requested from DB_i is produced by DB_i

materialized: $t^q_{i,der}$ is the time of the last update from DB_i that has been incorporated into the view.

In particular, then, the answer to query q posed at time t^q_{ask} will be $q(\nu(state(\vec{DB},\vec{t}^q_{der})))$.

In the materialized case, $t^q_{i,der}$ should not be used to measure staleness, because there may be occasions when a source database is not updated for a long period after $t^q_{i,der}$. Rather, we should look at the latest time after $t^q_{i,der}$ where the state of DB_i has not yet changed. We say that the query answer "reflects" the source database state of this time. Formally, for query invocation q and source database DB_i, $t^q_{i,refl}$ is the latest[1] time $t \leq t^q_{get}$ such that

$$state(DB_i,t') = state(DB_i,t^q_{i,der})$$

for all t' satisfying $t^q_{i,der} \leq t' \leq t$, For the virtual case, $t^q_{i,refl} = t^q_{i,der}$.

To illustrate these notions, consider a view containing a join of relations R and S, coming from source databases DB_R and DB_S. Figure 4 shows the relationship between the various times for a representative scenario involving with a query invocation q.

We define the (worst-case) *staleness* of DB_i in an integration environment involving multiple source databases $(DB_1,\ldots,DB_n)$, to be

$$\max\{\ t^q_{get} - t^q_{i,refl} \quad | \quad q \text{ is a possible query invocation, during normal operation of the environment }\}$$

The overall (worst-case) staleness is the maximum of these values over $i = 1,\ldots,n$.

[1] Recall that in this study, time is assumed to be discrete.

4.2 Selection views

We now consider the staleness of materialized and virtual selection views. We begin by developing formulas that characterize the staleness of different kinds of views and approaches that support those views. To this end, we introduce two additional time components:

update transfer time (UTT) the transfer time of the update from the source to the mediator,

view maintenance time (VMT) the time the mediator takes to propagate the update to the materialized view.

The staleness of a virtual selection view is equivalent to the answer transfer time:

$$Staleness_{select,virt} = ATT.$$

This is because the data used in the answer is considered to be fresh when it leaves the source database. Based on [HZ96], the staleness of a materialized view is

$$\begin{aligned} Staleness_{select,mat} = \ & ann\_delay + UTT \\ & + u\_hold\_delay_{med} \\ & + VMT + QET_{mediator} \end{aligned}$$

where ann_delay and $u\_hold\_delay_{med}$ are the announcement delay in the source (i.e., time between an update to the source and that update being announced to be mediator) and the update holding delay in the mediator (i.e., the time between receipt of an update and propagation into the view). We focus here on eager update reporting and eager update processing. In this case both ann_delay and $u\_hold\_delay_{med}$ are zero, and so the above formula can be simplified to:

$$Staleness_{select,mat} = UTT + VMT + QET_{mediator}.$$

Thus, the staleness of the materialized view corresponds to the sum of the time for transferring update from the source to the mediator, the time the mediator takes to propagate the updates to the materialized view, and the query execution time at the mediator. The above is the worst case, because we assume that there is an update at the source as soon as the answer of the query is produced.

Table 4 shows the staleness in milliseconds of both virtual and materialized selection views specified in the Base case (see Subsection 3.1). In this experiment, the materialized is staler than the virtual view. If we consider wide area network or higher query selectivity, the staleness of the virtual view may be worse. Figure 5(a) gives a bar chart for the numbers in tha table (not to exact scale).

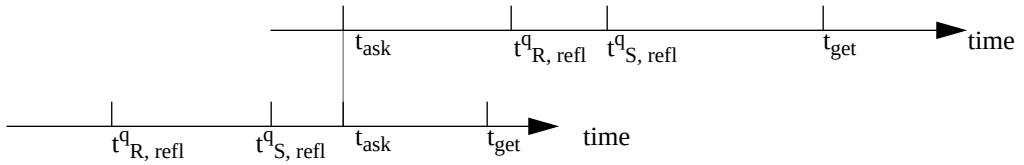


Figure 4: Relationship between the various times associated with a query invocation q

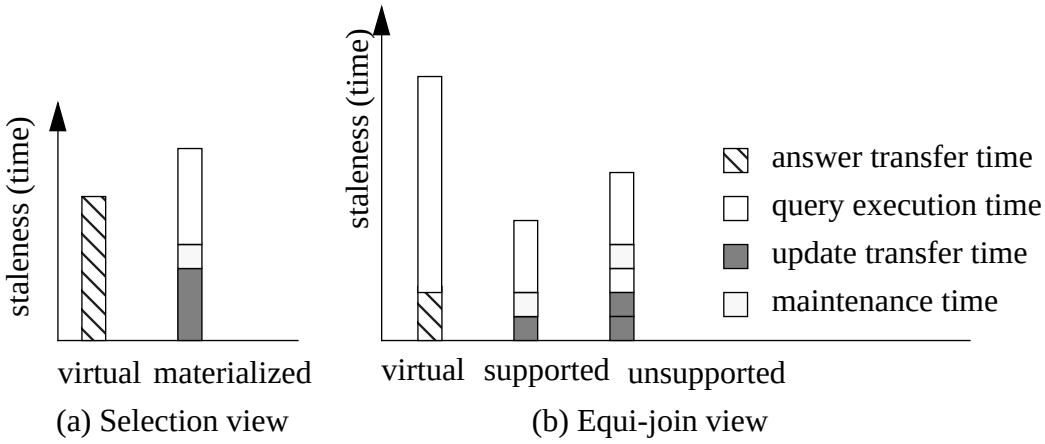


Figure 5: A chart of the staleness of a selection view and a join-view

time (ms)	virtual	materialized
ATT	243	
UTT		124
VMT		3
$QET_{mediator}$		301
staleness	**243**	**428**

Table 4: Staleness of virtual and materialized selection views in milliseconds

4.3 Join views

We first develop formulas that characterize the staleness of join views, and then present preliminary benchmarking results.

The formula for virtual, polling-based, join is:

$$Staleness_{join,virt} = ATT + QET_{mediator}.$$

Compared to $Staleness_{select,virt}$, $Staleness_{join,virt}$ includes the extra $QET_{mediator}$ that computes join of two relations polled from sources. For fully supported join, the formula is:

$$Staleness_{join,supported} = UTT + VMT + QET_{mediator}.$$

This is almost identical to $Staleness_{select,supported}$, except the VMT portion of the former takes a little more time, because update propagation to the join view is a more involved. As long as the updates are small, this extra time is almost negligible.

Turning to the unsupported materialized case, we have:

$$Staleness_{join,unsupported} = UTT + UTT + QET_{source} + VMT + QET_{mediator}$$

The first UTT is the time spent on sending the update from the source to the mediator, while the second UTT is the result of sending (a projection of) that update from the mediator to the other source database in order to compute (the reason for QET_{source}) the incremental update to the join view. The last two summands are analogous to those in the formula for the supported approach.

Table 5 gives the benchmarking results of the staleness of the join views. Figure 5(b) gives a bar chart for the numbers in tha table (not to exact scale). We observe that the main reason that the virtual view is much worse than the materialized views stems from the $QET_{mediator}$ portion. For the materialized views it is just a selection, but for the virtual view a join has to be performed. In the base case the join involves 10% selectivity of the two source relations, and so the join computation takes about three times as long as computing the selection. We further observe that the staleness of the

time (ms)	virtual	supported	unsupported
ATT	223		
UTT		124	124
UTT_2			124
QET_{source}			165
VMT		24	3
$QET_{mediator}$	2767	813	813
staleness	**2990**	**961**	**1229**

Table 5: Staleness of virtual and materialized join views in milliseconds

unsupported materialized view is generally greater than that in the supported case, because the former sends queries to the sources in order to perform view maintenance, which increases the view maintenance time.

The above experiment assumes a single selectivity (10%) of the query against the join view. Figure 6 shows the benchmarking results of the staleness of various join views with regards to different selectivities of the queries against the view. We observe that the staleness of the materialized view increases very slowly with higher selectivity. In contrast, the staleness of the two virtual views worsens dramatically, especially for complex join views.

The above experiments all assume eager update reporting/processing policies. What about a materialized view with a lazy/periodic maintenance policy? If moderate staleness can be permitted, then updates from the source databases can be batched, and typically the time between update transmission and propagation will be much larger than the data transfer time or update propagation time. As a result, under this policy the staleness can be estimated as the time between periodic updates. Suppose now that response time is to be minimized, and that staleness is a concern but does not need to be minimized. Figure 6 suggests a possible trade-off between staleness and system load: if there is high query selectivity then updates can be batched and propagated at periods of 10 or more seconds, while still obtaining freshness substantially better than the freshness of the virtual approach. This observation may also have implications in connection with the client-server architecture.

5 System Load

In this section, we investigate the system load caused by the various approaches for supporting data integration. We are mainly interested in four kinds of system load: mediator space usage, network traffic, mediator disk I/O, and source disk I/O. The space usage issue is quite straightforward: The materialized approaches require predictable amount of extra space in the mediator, based on the selectivity of the view definition is determined. We now consider the latter three kinds of system load in turn. We only consider join views here. The results can be easily extended to the selection views.

Based on the join algorithms described in Section 2, the amount of network traffic (NT) for the virtual, supported, and unsupported approaches can be estimated as follows (assuming uniform queries and updates):

(1) $NT_{join,virtual} = query\_rate * 2 * (query\_shipping + answer\_shipping)$

(2) $NT_{join,supported} = update\_rate * update\_size$

(3) $NT_{join,unsupported} = update\_rate * (update\_size + query\_shipping + answer\_shipping)$

Figure 7 shows the amount of network traffic for the three approaches as a function of query rate and source update rate. As indicated in Formula (1), the network traffic for the virtual approach is independent of the update rate, but it goes up when the query rate increases. The network traffic for the two materialized approaches is proportional to the update rate, but independent from the query rate because only local data (in the mediator) is accessed to answer the queries. The curve for the unsupported approach is steeper because a large volume of traffic is generated by the $answer\_shipping$ part (see Formula (3)).

Finally, we consider the processing load on the mediator and source databases. We estimate this by using the number of disk I/Os performed (see Figures 8 and 9). Again, in both the mediator and the source the amount of disk I/O for the virtual approach is independent from the source update rate, but it increases with the query rate. In the mediator, the disk I/O performed for the two materialized approaches is the result of answering queries and performing maintenance of materialized data. The number increases along with the increase of the update or query rate. The amount of disk I/O in the mediator for the unsupported approach is slightly lower than that for the supported approach. That is because part of the evaluation for the view maintenance for the former is performed in the source. This also explains why a considerable amount of disk I/O is performed in the source for this approach. The supported approach does not cause extra disk I/O in the source, because a smart log transfer manager, such as the one implemented in IBM DB2, can fetch the updates directly from the main memory at the source and send them to the mediator.

For a given query rate, an interesting number is the source update rate at which the mediator disk I/O for the virtual approach equals the mediator disk I/O for the supported (or unsupported) materialized case. Note that this crossing point moves to the right as the query rate increases. This suggests that in the case of heavily queried mediators, system load can be reduced by using materialization.

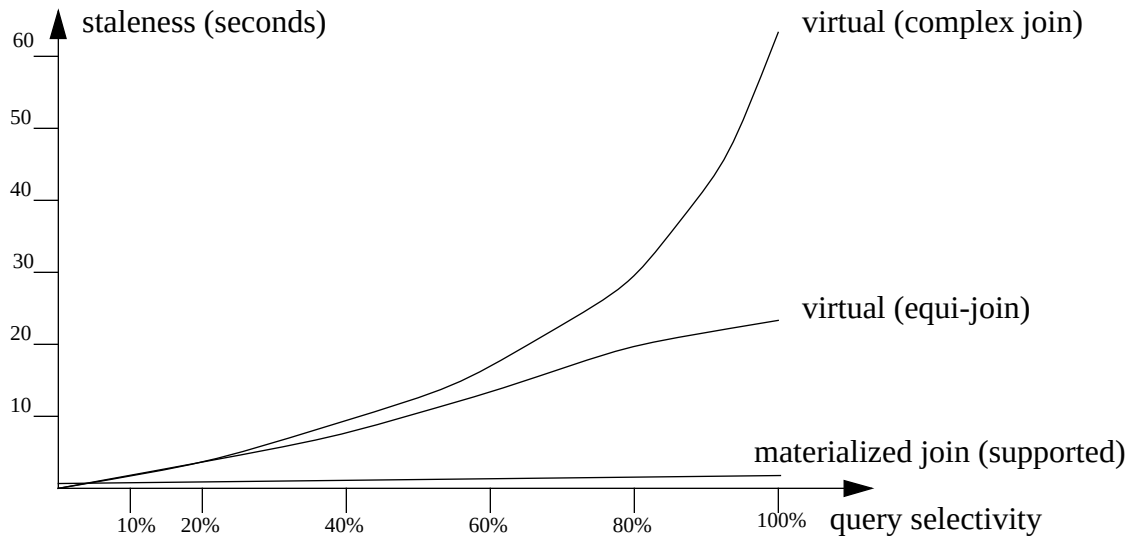


Figure 6: Staleness of join views with varying query selectivities queries against the view

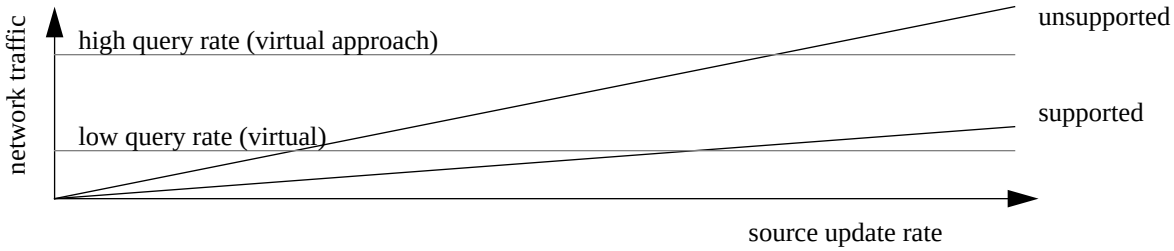


Figure 7: Network traffic

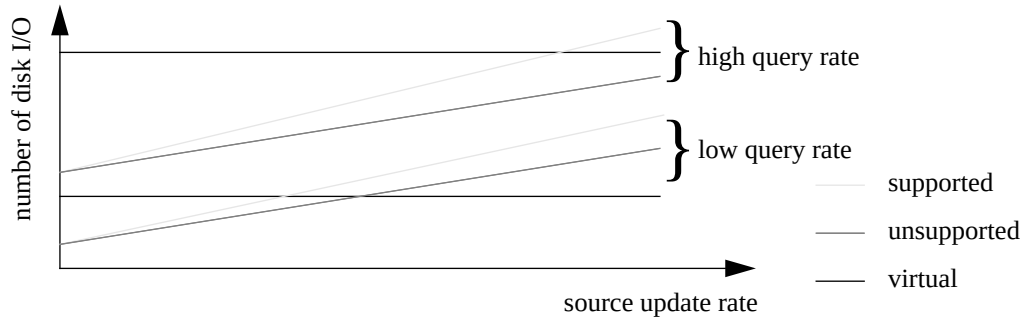


Figure 8: Number of disk I/O in the mediator

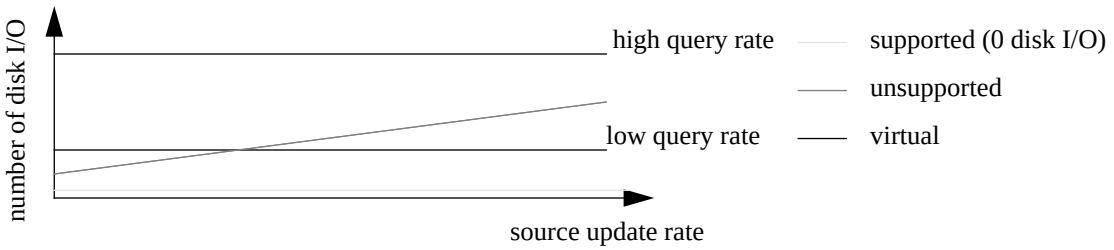


Figure 9: Number of disk I/O in the source

References

[ACHK93] Y. Arens, C.Y. Chee, C.N. Hsu, and C.A. Knoblock. Retrieving and integrating data from multiple information sources. *Intl. Journal of Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[BG81] P. A. Bernstein and N. Goodman. The power of natural semi-joins. *SIAM Journal on Computing*, 10(4):751–771, 1981.

[CDRS93] M. J. Carey, D. J. DeWitt, Joel E. Richardson and E. J. Shekita. Object and file management in the EXODUS extensible database system. In Proc. of Intl. Conf. on Very Large Databases, pages 91–100, 1986.

[DR93] A. Delis and N. Roussopoulos. Performance comparison of three modern DBMS Architectures. *IEEE Trans. on Software Engineering*, 19(2):120–138, 1993.

[FWW+] T. Finin, J. Weber, G. Wiederhold, et al. *DRAFT Specification of the KQML Agent-Communication Language.* Developed by the DARPA Knowledge Sharing Initiative External Interfaces Working Group, June 15, 1993.

[GHJ+93] S. Ghandeharizadeh, R. Hull, D. Jacobs, et al. On implementing a language for specifying active database execution models. In *Proc. of Intl. Conf. on Very Large Data Bases*, pages 441–454, 1993.

[GHJ96] S. Ghandeharizadeh, R. Hull, and D. Jacobs. Heraclitus: Elevating deltas to be first-class citizens in a database programming language. *ACM Trans. on Database Systems*, 1996. To appear. Available via anonymous ftp at ftp://ftp.cs.colorado.edu//users/hull/hera-tech94-revised.ps.

[GJM96] A. Gupta, H.V. Jagadish, and I.S. Mumick. Data integration using self-maintainable views. In *Proc. of Intl. Conf. on Extending Data Base Technology*, 1996.

[Han87] E. N. Hanson. A performance analysis of view materialization strategies. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 440–453, 1987.

[HZ96] R. Hull and G. Zhou. A framework for supporting data integration using the materialized and virtual approaches. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 481–492, 1996.

[T+90] G. Thomas et al. Heterogeneous distributed database systems for production use. *ACM Computing Surveys*, 22(3):237–266, September 1990.

[WHW89] S. Widjojo, R. Hull, and D. Wile. Distributed Information Sharing using WorldBase. *IEEE Office Knowledge Engineering*, 3(2):17–26, August 1989.

[Wie92] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, March 1992.

[ZGHW95] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 316–327, San Jose, California, May 1995.

[ZHK96] G. Zhou, R. Hull, and R. King. Generating data integration mediators that use materialization, 1996. *Journal of Intelligent Information Systems.* To appear. Available via anonymous ftp at ftp://ftp.cs.colorado.edu//users/hull/squirrel: materialization-JIIS.ps.