9th IEEE/ACM International Symposium on Cluster Computing and the Grid

Hierarchical Caches for Grid Workflows

David Chiu   Gagan Agrawal
Department of Computer Science and Engineering
The Ohio State University, Columbus, OH 43210, USA
{chiud,agrawal}@cse.ohio-state.edu

***Abstract*—From personal software to advanced systems, caching mechanisms have steadfastly been a ubiquitous means for reducing workloads. It is no surprise, then, that under the grid and cluster paradigms, middlewares and other large-scale applications often seek caching solutions. Among these distributed applications, scientific workflow management systems have gained ground towards mitigating the often painstaking process of composing sequences of scientific data sets and services to derive virtual data. In the past, workflow managers have relied on low-level system cache for reuse support. But in distributed query intensive environments, where high volumes of intermediate virtual data can potentially be stored anywhere on the grid, a novel cache structure is needed to efficiently facilitate workflow planning. In this paper, we describe an approach to combat the challenges of maintaining large, fast virtual data caches for workflow composition. A hierarchical structure is proposed for indexing scientific data with spatiotemporal annotations across grid nodes. Our experimental results show that our hierarchical index is scalable and outperforms a centralized indexing scheme by an exponential factor in query intensive environments.**

I. Introduction

For years, the scientific community has enjoyed ample attention from the computing society as a result of new and compelling challenges that it poses. These issues, which fall under the umbrella of data intensive scientific computing problems, are largely characterized by the need to access, analyze, and manipulate voluminous scientific data sets. High-end computing paradigms, ranging from supercomputers to clusters and the heterogeneous grid, have lended well to middlewares and applications that address this set of problems [13].

Among these applications, workflow management systems have garnered considerable interest because of their ability to manage a multitude of scientific computations and their interdependencies for deriving the resulting products, known as *virtual data*. Although a substantial amount of effort in this area has been produced, great challenges for grid-enabled scientific workflow systems still lie ahead. Recently, Deelman et al. outlined some of these challenges [7]. Among them, data reuse is one of particular interest, especially in the context of autonmous systems. While questions on how best to identify the existence of intermediate data as well as determining their benefits for workflow composition remain open, the case for providing an efficient scheme for intermediate data caching can certainly be made.

Historically, caching mechanisms have been employed as a means to speed up computations. Distributed systems, including those deployed on the grid, have relied on caching and replication to maximize system availability and to reduce both processing times and network bandwidth consumption [4]. In the context of scientific workflow systems, we could envision that intermediate data generated from previous computations could be stored on an arbitrary grid node. The cached virtual data may then be retrieved if a subsequent workflow calls for its use.

To exemplify, consider Figure 1, which depicts a workflow manager that is deployed onto some scientific (in this case, geospatial) data grid. In this particular situation, a workflow broker maintains an overview of the physical grid, e.g., an index of nodes, data sets, services, as well as their inter-relationships. The broker, when given a user query, generates workflow plans and schedules their execution before returning the virtual data result back to the user. Focusing on w_{tj}, this workflow initially

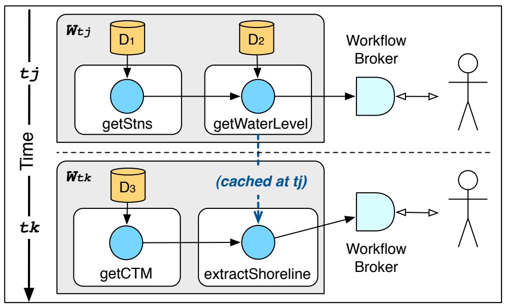


Fig. 1. Example Workflow Sequence

invokes getStns(), which returns a list of water gauge stations close to some area of interest. This list is input to another service, getWaterLevel(), which queries each station from the input list for their readings at the desired time. After a series of computations, getWaterLevel() eventually produces the desired virtual data: average water level for some given region and time. Now, let's assume a second query is submitted at some later time $tk > tj$, involving shoreline extraction for those same time and region. The first half of w_{tk} invokes getCTM() to identify and retrieve spatiotemporally relevant CTM data. This is input into extractShoreline(), which also requires the water level. Having been processed earlier at tj, the water level redundant, and w_{tk}'s execution time can be reduced if our system can efficiently identify whether this virtual data already exists.

The workflow redundancy exhibited above might seem a bit improbable in a spatiotemporal environment where space and time are vast and users' interests are disparate. Such situations, however, are not absent from *query intensive* circumstances. For instance, (i) an unexpected earthquake might invoke an onslaught of similar queries issued for a specific time and location for examination and satisfying piqued curiosities. (ii) Rare natural phenomena such as a solar eclipse might prompt a group of research scientists with shared interests to submit sets of similar

978-0-7695-3622-4/09 $25.00 © 2009 IEEE
DOI 10.1109/CCGRID.2009.10

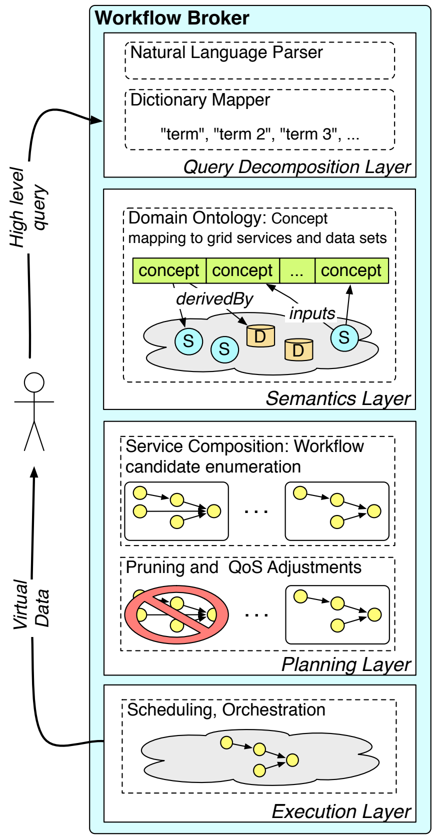
experiments with repeated need for some intermediate data. Without virtual data caching, a workflow system may not be able to adequately cope with the sudden surge in queries for the amount of data movement and analysis necessary. Managing a cache under these situations, however, is met with certain difficulties. In this paper, we address approaches in handling several technical challenges towards the design of a grid based cache-sensitive workflow composition system. These challenges, and our contributions, include:

- *Providing an efficient means for identifying cached virtual data* — Upon reception of a user query, our automatic workflow composition system immediately searches for paths to derive the desired virtual data. It is within this planning phase that relevant virtual data caches should be identified, extracted, and composed into workflows, thereby superseding expensive service executions and large file transfers. Clearly, the cache identification process must only take trivial time to ensure speedy planning.
- *Dealing with high-volumes of spatiotemporal data* — Large amounts of intermediate virtual data can be cached at any time in a query intensive environment. But scientific qualifications, such as spatiotemporality, mixed in a potentially high update environment will undoubtedly cause rapid growth in index size. To this end, we have proposed an efficient index structure with an accompanying victimization scheme for size regulation.
- *Building a scalable system* — A large distributed cache system should leverage the grid's versatility. Our cache structure is designed in such a way as to balance the index among available nodes. This consequently distributes the workload and reduces seek times as nodes are introduced to the system.

In the ensuing sections, we describe our proposed advances towards solving the aforementioned challenges. Initially, in Section II, we revisit our previously developed workflow system. In Section III, we present the integration of our current workflow with the proposed structures and access methods for indexing intermediate virtual data. Experiments were designed to gauge the performance and scalability of our system. We showed that the caches improved the running times of our geospatial case study applications by an average factor of 3.5. We were also able to justify our proposed index structure through a display of graceful scalability to grid-scale networking environments. These and other results are presented in Section IV. Finally, in Sections V and VI, we respectively discuss related efforts in this area and conclude with thoughts on future opportunities.

II. Background and Overview

Our system is a grid-enabled workflow composition engine. Although our examples are in the context of geospatial workflows, the underlying system framework is general. Here, we only give a brief overview of the necessary background of our system's architecture (nuanced descriptions found in [5], [6]). Figure 2 shows the system architecture. It is composed of four independent layers, i.e., the design and implementation of each layer can be replaced without affecting the others as long as some system-specified protocols are met. From a high-level perspective, the system can be viewed as a *workflow broker:* As users submit queries to the system, the broker plans and executes the workflows involved in deriving the desired virtual data while hiding such complexities as workflow composition, domain knowledge, QoS optimization, and grid communications from the user.



Workflow Broker
Natural Language Parser
Dictionary Mapper
"term", "term 2", "term 3", ...
Query Decomposition Layer
High level query
Domain Ontology: Concept mapping to grid services and data sets
concept
concept
...
concept
derivedBy
inputs
S
D
Semantics Layer
Service Composition: Workflow candidate enumeration
Pruning and QoS Adjustments
Planning Layer
Virtual Data
Scheduling, Orchestration
Execution Layer

Fig. 2. System Architecture

The user queries through the broker's *Query Decomposition Layer*, which employs a natural language parser, StanfordNLP [18], to decompose the query into keywords and dependencies. Using WordNet [9], we reduce the keyword set to a manageable size. This reduced keyword set is mapped to concrete concepts within the domain ontology resident in the subsequent layer. The *Semantics Layer* maintains an active list of available nodes, services, data sets, and their metadata. Services are annotated with WSDL and nodes with such information as their location, relative bandwidths, etc. Data must also be cataloged. In the geospatial domain, our data sets are coupled with CSDGM (Content Standard for Digital Geospatial Metadata) [10]. This annotation, a standard in the geographic community, provides spatiotemporal information in addition to conventional metadata such as the author, date of creation, etc. These system-level semantics alone are not sufficient to handle all complexities in autonomous workflow planning. There is an additional requirement for domain-level semantics. For example, there is a need for the system to understand how "water levels" in general are derived using some existing data sets, services, or combinations of both.

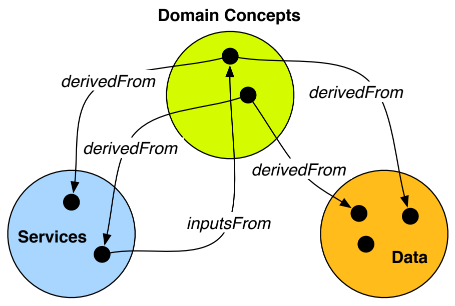


Fig. 3. Ontology for Domain Description

Algorithm 1 WFEnum($target$, $immData$)

```
1: $W \leftarrow \emptyset$
2: /* $B$ denotes the set of all data/service elements that can be used to
   derive $target$ concept */
3: $B \leftarrow target$.derivedFrom()
4: for all $\beta \in B$ do
5:   if $\beta$ is a data set then
6:     $W \leftarrow \beta$.getData()
7:   else
8:     /* $\beta$ involves a service invocation */
9:     $P_\beta \leftarrow \beta$.getServiceParams()
10:    $W_s \leftarrow \emptyset$
11:    for all $p \in P_\beta$ do
12:      $W_s \leftarrow W_s \cup$ WFEnum($p.concept$, $immData$)
13:    end for
14:    for all $pm \in W_s$.crossProduct() do
15:      $W \leftarrow srvc(\beta, pm)$
16:    end for
17:  end if
18: end for
19: return $W$
```

These relationships can be specified through a simple ontology, displayed in Figure 3. The ontology makes it easy to clearly indicate which services and data sets are responsible for deriving a specific domain concept. A service's parameters must also refer back to concepts for modeling their meaning. The separation of semantics from other system layers is to provide a simple means for switching context. To enable our system for another domain, for instance, we simply replace the ontology.

With a semantics framework in place, it is now possible to support workflow planning. Our algorithm, called workflow enumeration (WFEnum), takes as input the targeted domain concept and a set of concept=value pairs representing immediate data. These, of course, can be assumed to be passed in the expected form from the previous layers. The WFEnum algorithm is a modification of Depth-First Search[†], shown in a simplified version, in Algorithm 1. The goal is to enumerate a set W of all workflow candidates capable of deriving the user query. Working off the ontology, WFEnum starts by first examining the query's targeted concept. By following each of its *derivedFrom* relationships, it constructs an independent workflow containing the services or data sets to which the link connects. If the *derivedFrom* element is a data set, then the base case is reached, and the data set is directly added to W (Lines 5-6). The counterpart is the case where the *derivedFrom* element is a service, starting with (Line 7). In this case, it scans through the service's parameters and derive each through a recursive invocation of WFEnum on the parameter's corresponding concept. W_s contains an intermediate workflow candidate set for the service at hand. This set is then crossed-product to generate all possible combinations of each parameter's derivation. Once coupled with the service call, workflows are added to W as possible candidates. After all paths have been visited for the original target concept, W is returned and passed down to the QoS adapation component.

Our system optionally allows user preferences in terms of time allowed and virtual data accuracy. With respect to these two constraints, our system enables an implementation of bi-criteria optimization that is based on a system of cost models. Each workflow's predicted time and accuracy are computed on the fly as workflows are enumerated. Candidates are immediately pruned based on the a priori principle if QoS scores do not meet user specifications. The problem with this pruning rule is that potential candidates might be eliminated too aggressively. For instance, a user may issue a query well-known for its 4 hour processing time, but is willing to surrender to some marginal errors in the resulting virtual data, if it could instead be answered within a half hour. Similarly, another user might prefer the fastest query time as long as it produces an 80+% accurate answer. Our system can flexibly meet these types of constraints through invoking online reduction/compression on data sets [6]. After this procedure, the pruned, or QoS-adapted set of workflows, is sent to the *Execution Layer* for processing, and the resulting virtual data is finally returned back to the user.

[†]In [5], we prove the WFEnum $\rightarrow$ DFS reduction and give an in-depth analysis of its complexity.

III. Enabling a Fast Distributed Cache

Workflows, in general, involve a series of service executions to produce some set of intermediate virtual data used to input into the next set of services. Caching these read-only intermediate results would clearly lead to significant speed up, particularly when replacing long running services with previously derived results. Returning back to Figure 2, the cache system is logically positioned in both the Planning and Execution Layers. Much of the challenges in its implementation is tied directly to the Planning Layer. For one, the existence of previously saved virtual data must be quickly identified so as to amortize the cache access overhead in the workflow enumeration phase. At first glance, it would then seem straightforward to place the virtual data cache directly on the broker node. Several issues, however, argue against this justification:

1) Services are distributed onto arbitrary nodes within the grid. Centralizing the cache would imply the need to transfer virtual data to the broker after each service execution. Moreover, accessing the cache would further involve virtual data transfer from the broker to the node containing the utilizing service. This would lead to an increase in network traffic on the broker, which should be avoided at all costs.
2) A centralized broker cache would scale poorly to large volumes of cached virtual data. Due to the nature of our spatiotemporal virtual data, multidimensional indices (e.g., R-Trees, its variants, and others [14], [24], [3]) can typically

be employed. Some issues are at stake: (i) Cached virtual data are read-only. In a high-insertion, query intensive environment, a centralized multidimensional index can quickly grow out of core [15]. (ii) To solve this issue, a cache replacement mechanism would be needed to contain the index within memory despite the fact that less virtual data can be tracked.

In hopes of alleviating the challenges outlined above, we introduce a system of hierarchically structured caches, shown in Figure 4. Again, the existence of a cached result must be known at planning time to ensure speedy enumeration. For the Planning Layer to access this information efficiently, it is unavoidable that some form of cache index must still exist on the broker with the caveat being that its size must be regulated. The *broker index* is

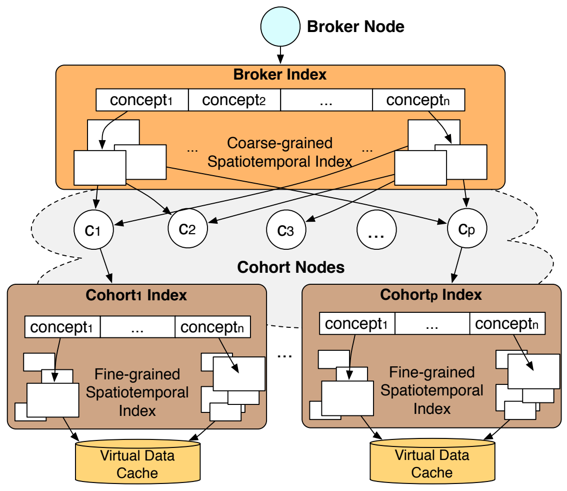


Fig. 4. Hierarchical Index

organized in two tiers: (i) A table of domain concepts (specified within the Semantics Layer's ontology) summarizes the top tier. Placing concepts at the top enables the search function to prune significant portions of the index prior to initiating the costly spatiotemporal search. (ii) In the bottom tier of the broker index, each concept maintains a distinct spatiotemporal index tree. In each tree we want its spatiotemporal granularity to be coarse. By broadening the span of time and spatial coverage that each virtual data element could hash into, we can dramatically reduce the broker index's size and thus reduce hit/miss times. Each broker index record contains pointers to any number of grid nodes, i.e., cohorts, that might contain the desired information.

A *cohort index* exists locally on each cache-enabled node in the grid. Its structure is not unlike that of the broker index, with the only difference being that it maintains a fine-grained spatiotemporal index tree. The logic is that, if enough nodes join the rank of cohorts, then each node can manage to cover increasingly finer spatiotemporal details. Moreover, the overall index size and load is balanced and shared. Each cohort index record contains the location of the virtual data on the local disk. Together, the cohorts represent a massive distributed spatiotemporal index.

Direct consequences of this hierarchical structure are the hit and miss penalties. While recognizing a miss is trivially contained within the broker, a hit cannot be fully substantiated until hits on the cohort level are reported. Thus, three responses are possible: fast miss, slow miss, hit (slow). One of the design goals is to support our hypothesis that, in query intensive environments where centralized indices can quickly grow out of core, hits/misses can be realized significantly faster on the hierarchical index despite the overhead of cohort communications. This claim is later verified in Section IV.

A. Bilateral Cache Victimization

The cost of maintaining a manageable broker index size is the ambiguity that leads to false broker hits (followed by cohort misses). With a large enough spatiotemporal region defined in the broker, only a small hash function can be managed. This means that a true miss is only realized after a subsequent miss in the cohort level. Keeping a finer grained broker index is key to countering false broker hits. But in a high-insert environment, it is without question that the index's size must be controlled through victimization schemes, e.g., LRFU [19]. Because of their direct

Algorithm 2 BrokerVictimization($brkIdx$, $V[\ldots]$, ϕ, τ)
```
1: while $\phi > \tau$ do
2:     /* $v$ is the victimized region key */
3:     $v \leftarrow V$.pop()
4:     $record \leftarrow brkIdx$.get($v$)
5:     /* broker records hold list of cohorts that may contain cached
       virtual data */
6:     /* broadcast delete to associated cohorts */
7:     for all $cohort \in record$ do
8:         $cohort$.sendDelete($v$)
9:     end for
10:    $brkIdx$.delete($v$)
11:    $\phi \leftarrow \phi - 1$
12: end while
```

ties, a broker record's victimization must be communicated to the cohorts, which in turn, deletes all local records within the victimized broker region. Cohort victimization, on the other hand, is not as straightforward. As each node can have disparate replacement schemes, a naïve method could have every cohort periodically send batch deletion requests to the broker. The broker deletes a region once it detects that all cohort elements have been removed from that entry. But this method is taxing on communications cost. To cut down on cost, we propose the following bidirectional scheme: the top-down Broker Victimization and the bottom-up Cohort Victimization.

Broker Victimization (Algorithm 2) takes as input the broker index, $brkIdx$, a queue of victims, $V[\ldots]$, the current record size, ϕ, and the record size threshold, τ. The algorithm is simple: as broker records are deleted to regulate index size back to τ, each involved cohort node must be communicated to delete its own records within the victimized region. This is repeated until ϕ is regulated down τ. The selection of an effective τ is largely dependent on system profiles (e.g., physical cache and RAM capacity, disk speed, etc), and can take some trial-and-error. For instance, we show in the experimental section that τ appears to be between 2 and 4 million records on our broker, which uses 1GB of RAM.

In solving for the complexity of Broker Victimization, we let C denote the set of cohorts in any hierarchical index. For some cohort node $c \in C$, we also define $t_{net}(c)$ to be the network transfer time from the broker to c. Finally, if we let $n = \phi - \tau$ be the amount of to-be-victimized records, the total time taken for Broker Victimization, $T_{bvic}(n)$, is:

$$T_{bvic}(n) = \sum_{i=1}^{n} (\max_{j=1}^{|C_i|}(t_{net}(c_j)) + \delta)$$

where $|C_i|$ denotes the number of cohorts that needs to be communicated to ensure the victimization of record i and δ is some trivial amount of local work on the broker (e.g., victim deletion). If we further let the slowest broker-to-cohort time be called t_m, i.e.,

$$t_m = \max_{c \in C}(t_{net}(c))$$

then the worst case bound is $T_{bvic}(n) = O(n(|C|t_m + 1))$.

Because the overall time is inevitably dominated by cohort communications, we propose an asynchronous version which minimizes $|C|$ to 0. On behalf of the a priori principle: When broker records are removed, it implies that a multitude of cohort records has also not been recently accessed. Eventually, regardless of each cohort's individual replacement scheme, the unused records will be evicted due to its absence from the broker index. In effect, cohort communication can essentially be omitted, reducing the algorithm to $O(n)$, or an amortized $O(1)$ depending on the frequency of its invocation and due to the triviality of the constant time δ. This, of course, is at the expense of each cohort having to maintain some amount of deprecated records.

When used alone, Broker Victimization is insufficient. If only a few elements exist in the broker's larger regions, the entire coarse-grained record must still be kept while less frequently used records in cohort indices might have already been evicted in their own victimization schemes. This leads to an inconsistency between the two indices and causes false broker hits. To handle this issue, we employ the Cohort Victimization scheme (not shown due to space limitations). Each cohort maintains a copy of its own relevant subset of broker's spatiotemporal coverage. When a cohort victimizes a record, an eviction message is sent to the broker if region which empasses the victim is now empty. Upon reception of this message, the broker removes the pointer to the evicted cohort node from the indexed element. Only after all cohort pointers have been emptied from that broker record does the broker delete the respective region.

B. Fast Spatiotemporal Indexing

When facing query intensive environments, frequent cache index updates must be anticipated. We utilize a slightly modified version of the B^x-Tree [16] for fast spatiotemporal indexing. Originally proposed by Jensen et al. for indexing and predicting locations of moving objects, B^x-Trees are essentially B+Trees whose keys are the linearization of the element's location via transformation through space filling curves. The B^x-Tree further partitions its elements according to the time of the update: Each timestamp falls into a distinct partition index, which is concatenated to the transformed linear location to produce the record's key. The appeal of this index lies in its underlying B+Tree structure. Unlike most high dimensional indices, B+Trees have consistently been shown to perform exceptionally well in the high update environments that query intensive situations pose. But since B+Trees are intended to capture 1-dimensional objects, the space filling curve linear transformation is employed.

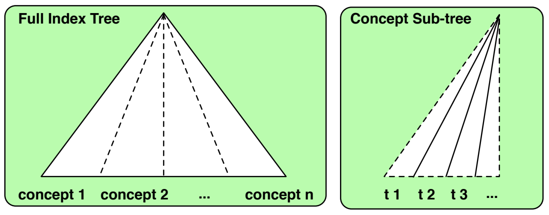


Fig. 5. A Logical View of Our B^x-Tree

In the B^x-Tree, space filling curves (a variety of curves exist; the Peano Curve is used in our implementation) [21] are used to map object locations to a linear value. In essence, these curves are continuous paths which visit every point in a discrete, multidimensional space exactly once and never crosses itself. The object's location, once mapped to a point on the space filling curve, is concatenated with a partition indexing the corresponding time.

Since our goal is not indexing moving objects and predicting their locations in present and future times, we made several adjustments to suit our needs. First, since the B^x-Tree tracks moving objects, their velocity is captured. In our implementation, we can simply omit this dimension. Second, our timestamps are not update times, but the physical times relevant to the virtual data. Finally, recall from Figure 4, that the notion for the concept-first organization for the broker and cohort indices is a means to provide fast top level pruning. In practice, however, maintaining a separate spatiotemporal index per concept is expensive. We propose an alternate approach: We also linearize the domain concepts by mapping each to a distinct integer value and concatenating this value to the leftmost portion of the key. By attaching binary concept mappings to the most significant bit portions, we logically partition the tree into independent concept sections, as shown in Figure 5. In the right side of the figure, we focus on a concept's sub-tree; each sub-tree is further partitioned into the times they represent, and finally, within each time partition lie the curve representations of the spatial regions.

We manipulate the key in this fashion because the B+Tree's native insertion procedure will naturally construct the partitioning without modifications to any B+Tree structures. This, due to the B+Tree's popularity, allows the B^x-Tree to be easily ported into existing infrastructures. The leftmost concatenation of concept maps also transparently enables the B+Tree search procedure to prune by concepts, again without modification of B+Tree methods. To clarify, if a virtual data pertaining to concept k is located in (x, y) with t being its time of relevance, its key is defined as the bit string:

$$key(k, t, o) = [k]_2 \cdot [t]_2 \cdot [curve(x, y)]_2$$

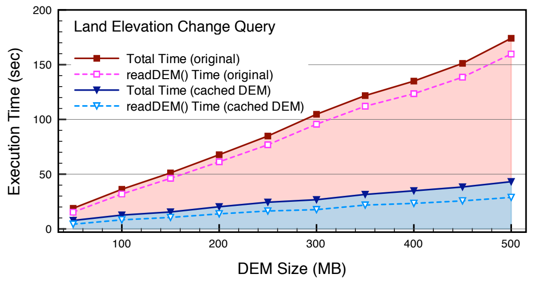
where $curve(x, y)$ denotes the space filling curve mapping of (x, y), $[n]_2$ denotes the binary representation of n, and $\cdot$ denotes binary concatenation.

IV. Experimental Results

In this section, we present an evaluation of our cache-enabled workflow system. In our grid environment, the broker node is a Linux machine running Pentium IV 3Ghz Dual Core with 1GB of RAM. The broker connects to a cluster of cohort nodes on a 10MBps link. Each cohort node runs dual processor Opteron 254 (single core) with 4GB of RAM. The cohort cluster contains 64 nodes with uniform intercluster bandwidths of 10MBps.

First, we pit our system against two frequently submitted geospatial queries to show the benefits of intermediate result caching. These are, Land Elevation Change="return land elevation change at (x, y) from time u_{old} to time u_{new}" and Shoreline Extraction= "return shoreline for (x, y) at time u."

To compute the Land Elevation Change query, a readDEM() service is used to identify and extract Digital Elevation Model (DEM) files into virtual objects corresponding to the queried time and location. This service is invoked twice for extracting DEMs pertaining to u_{old} and u_{new} into compressed objects. The compressed DEM objects are passed on to finish the workflow. We measured the overall workflow execution time for various sized DEMs and displayed the results in Figure 6 (top). The



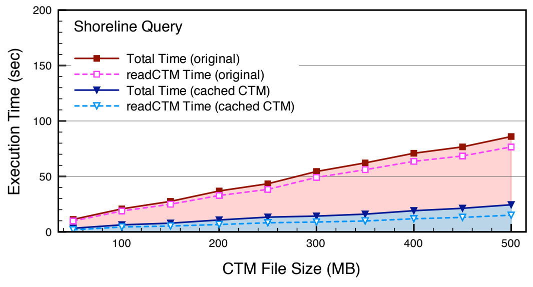
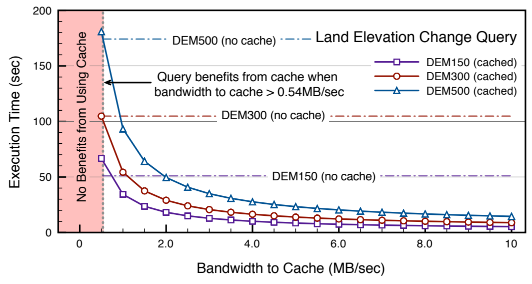


Fig. 6. Effects of Caching on Reducing Workflow Execution Times

solid-square line, denoted *Total Time (original)*, is the total execution time taken to process this query without the benefits of caching. The dotted-square line directly underneath, denoted *readDEM() Time (original)*, shows the time taken to process the two readDEM() calls. Regardless of DEM size, readDEM() dominates, on average, 90% of the total execution time. If the intermediate DEM objects can be cached, the calls to readDEM() can simply be replaced by accesses to the compressed DEM objects in the cache. The triangular lines in Figure 6 (top) indicate the benefits from using the cache. Due to the reduction of readDEM() to cache accesses, the same workflow is computed in a drastically diminished time. The average speed up that caching provides over the original workflow is 3.51.

The same experiment was repeated for the Shoreline Extraction query. In the workflow corresponding to this query, a readCTM() service stands as its dominant time factor. Not unlike readDEM(), readCTM() extracts Coastal Terrain Models (CTM) from large data sets into compressed CTM objects. As seen in Figure 6 (bottom), we consistently attain average speed ups of 3.55 over the original, cache-less executions, from utilizing cached versions of CTM objects.



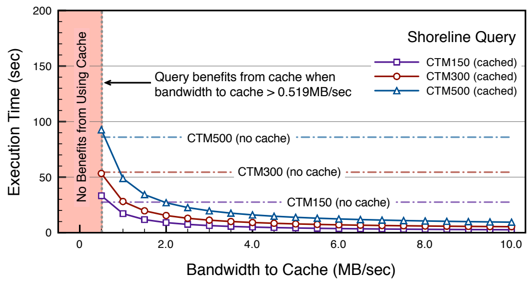


Fig. 7. Varying Bandwidths-to-Cache

The next set of experiments looks at the effectiveness of our cache system over heterogeneous networking environments, expected in the grid. We execute the previous workflows, this time with three fixed settings on virtual data size. Here, an advantage of virtual data sets is shown. Recall readDEM() and readCTM() both read large files into compressed objects. For original DEM and CTM files of size 150MB, 300MB, and 500MB, their respective virtual object sizes are 16.2MB, 26.4MB, and 43.7MB. This is fortunate, as our system only needs to cache the compressed objects. In these experiments, we are interested in the point in broker-to-cache bandwidth where it becomes unreasonable to utilize the cache because it would actually be faster to execute the workflows in their original formats. Our hope is that the cache will provide enough speed up to offset the overhead induced by slow links. Figure 7 displays the results for this experiment on Land Elevation Change (top) and Shoreline Extraction (bottom). Among the three fixed DEM/CTM sizes (150MB, 300MB, and 500MB), we found that we will on average attain speed ups over broker-to-cache links greater than 0.54MBps for the Land Elevation Change workflow and 0.519MBps for Shoreline Extraction. In a typical scientific grid or cluster environment we believe that it is reasonable to assume the existence of average bandwidths either at or above these values. Still, one can see how, by monitoring network traffic on the cache link and building a model around the results of these experiments, our system can decide whether or not the cache should be utilized for workflow execution. The bandwidth monitor, however, is not yet implemented in our system.

The last set of experiments provide insight into aspects of scalability. First, we investigate average seek times between our hierarchical structure and a centralized index. The centralized index is equivalent to a single broker index without cohorts. To facilitate this experiment, we similated a query intensive environment by inserting an increasing amount of synthetic records into our index. In the centralized structure, a massive B^x-Tree is used to index the entire virtual data cache. Because cohort communications is avoided, we should expect far faster seek times for smaller index sizes. In Figure 8, the centralized index's seek time is illustrated by the lone solid line and follows the left y-axis.

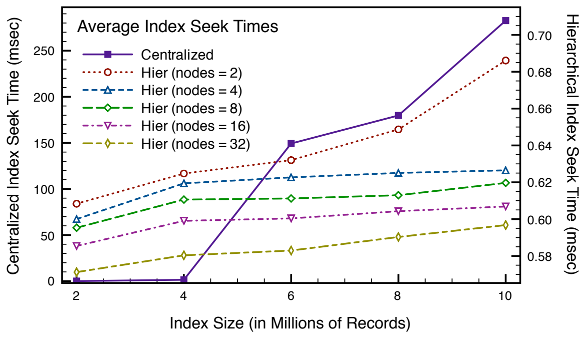


Fig. 8. Broker and Cohort Seek Times

Not shown clearly in the graph, the centralized index queries are 0.024 msec and 1.46 msec respectively for 2×10^6 and 4×10^6 records. This supports the claim that the centralized version outperforms the hierarchical index if its record size is relatively small. But as the index grows into disk memory (in our case, around 6×10^6 records), queries on this index are subject to an exponential slowdown.

The hierarchical index was designed to elude this problem by partitioning the index's bulk into manageable parts shared by cohorts. The downside to this model, however, is that a hit in the broker index requires a subsequent hit on the cohort level. Recall that a broker (fast) miss requires no cohort communications, and are thus omitted from our experiments because we are only interested in the penalties caused by cohort communications. The results, guided on the right-hand y-axis, tells us two things. First, the hierarchical scheme for small indices (less than 6×10^6 records) is expectedly slower than a centralized index. However, unlike the centralized scheme, average seek times do not suffer an exponential slowdown as records increase beyond 6×10^6 records because the load is split across the cohorts.

The second observation is that, as cohort nodes are added, we notice a slight but consistent speed up in average seek times. This speed up is due to the cohorts being assigned smaller sets of records, and thus, faster querying. This is made clear in Figure 9, in which we used an index containing 10 million records and show its reduction once partitioned into $2, 4, \ldots, 32$ cohorts. While the decline in cohort size is obvious due to index partitioning, the subtle increase in the broker index size can be

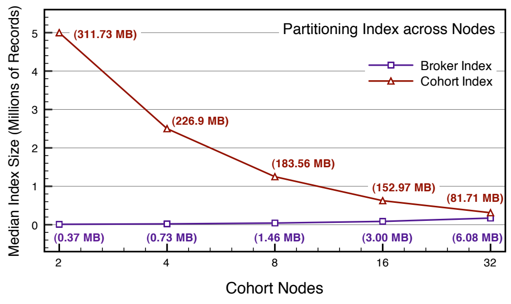


Fig. 9. Broker and Cohort Index Sizes

explained by increased granularity. For example, when only two cohorts exist, the broker's spatiotemporal regions can only be split into one of the two cohorts. Most of the broker's records, in this case, simply hashes into the same regions, resulting in a small broker index. But as cohort nodes increase, the broker index becomes larger because records can now be hashed into far more regions — an increase in broker granularity. In summary, our results first show the potentials for intermediate data caching in scientific workflow systems. We show that our hierarchical cache scheme is effective even in low to medium bandwidth environments often resident in heterogeneous grid environments. Most importantly, our results provide evidence supporting our case against strictly using a centralized broker index in a query intensive environment. The experiments reinforces our claim that a hierarchical structure is both efficient and scalable to grid-scaled systems. Moreover, much more virtual data can be indexed using this strategy over an aggressively victimized centralized index.

V. Related Works

Many grid workflow systems have been developed to support the growing interest in data intensive computing for large scale scientific applications [22], [20], [8], [2]. These systems enable automatic scheduling and execution of workflows on the order of thousands of processes (or activities). Akin to our own efforts, Askalon [23] implements a domain oriented framework to separate data meaning and representation to hide such complexities of data types from users when composing workflows. Our system differs from the above in that it answers high level user queries while abstracting all complexities of workflow composition from the user. We do this through a collaborative mixture of layers of semantics and workflow planning.

In the direction of data reuse, distributed file caching and replication architectures have long been established as an effective means for reducing processing times for compute and data intensive processes [26], [4], [27]. Meanwhile, recording provenance, or detailed information representing the procedure from which virtual data are derived, became very popular for helping users accurately reproduce certain results. Chimera [11] was an early endeavor on building technologies for cataloging virtual data provenance. Recognizing its benefits, provenance technologies also emerged in workflow systems [1], [17], [25]. Recently, Shankar and Dewitt integrated distributed virtual data caches with previous jobs' metadata (essentially a form of provenance) in their Condor-based scientific workflow manager, DAG-Condor [12].

Their goals of exposing distributed, preprocessed virtual data for the purposes of memoization are consistent with our objectives. In DAG-Condor, workflow plans are declarative: Users submit a job file that specifies input/output files, variables, dependencies, etc. DAG-Condor thus assumes virtual data used for input/output can be indexed on checksums of job histories. This user-guided nature marks a fundamental contrast between our systems.

Because our system automates workflow composition, our indexing scheme must be robust enough to correctly and efficiently identify virtual data during the planning process. To automate this process, domain level semantics must be injected to our cache identification mechanism. Thus, our index is not only physically structured across the grid to quickly identify relevant cache locations, but the individual indices themselves are strongly tied to the scientific domain with knowledge and spatiotemporal connotations. To the best of our knowledge, our system is the first to utilize a hierarchical and knowledge-enabled cache scheme in a automatic workflow composition framework.

VI. Conclusion and Future Directions

In this paper, we proposed a novel approach towards an autonomous, cache-enabled, grid workflow system. We presented the integration of our existing system with a hierarchical spatiotemporal indexing scheme for capturing preexisting virtual data. To maintain manageable indices and cache sizes, we set forth a bilateral distributed victimization scheme. To support a robust spatiotemporal index, a domain knowledge-aware version of the B^x-Tree was implemented. Our experimental results show that, for two frequently submitted geospatial queries, the overall execution time improved by a factor of over 3.5. The results also suggested that significant speed ups could be achieved over low to medium bandwidth environments. Lastly, we showed that our indexing scheme's search time and index size can scale to the grid.

As the scientific community continues to push for enabling mechanisms that support compute and data intensive applications, grid workflow systems will experience no shortage of new approaches towards optimization. We are currently investigating a generalized version of our hierarchy. In large and fast networks, it may be worth maintaining multiple broker levels with increasingly granular regions before reaching the cohorts level. In this framework, as nodes are added or removed, a evolution of broker splits and cohort promotions will involve a detailed study of the effects of index partitioning and restructuring.

Acknowledgments

This work was supported by NSF grants 0541058 and 0619041. The equipment used for the experiments reported here was purchased under the grant 0403342.

References

[1] I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance collection support in the kepler scientific workflow system. pages 118–132. 2006.

[2] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows, 2004.

[3] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[4] W. Bethel, B. Tierney, J. lee, D. Gunter, and S. Lau. Using high-speed wans and network data caches to enable remote and distributed visualization. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, Dallas, TX, USA, 2000.

[5] D. Chiu and G. Agrawal. Enabling ad hoc queries over low-level geospatial datasets. Technical report, The Ohio State University, 2008.

[6] D. Chiu, S. Deshpande, G. Agrawal, and R. Li. Cost and accuracy sensitive dynamic workflow composition over grid environments. In *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (Grid'08)*, 2008.

[7] E. Deelman and A. Chervenak. Data management challenges of data-intensive scientific workflows. In *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 687–692, Washington, DC, USA, 2008. IEEE.

[8] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. C. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[9] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. The MIT Press, 1998.

[10] Metadata ad hoc working group. content standard for digital geospatial metadata, 1998.

[11] I. T. Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *SSDBM '02: Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, pages 37–46, Washington, DC, USA, 2002. IEEE Computer Society.

[12] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC)*, pages 7–9, San Francisco, California, August 2001.

[13] L. Glimcher and G. Agrawal. A middleware for developing and deploying scalable remote mining services. In *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 242–249, Washington, DC, USA, 2008. IEEE Computer Society.

[14] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.

[15] C. S. Jensen. Towards increasingly update efficient moving-object indexing. *IEEE Data Eng. Bull*, 25:200–2, 2002.

[16] C. S. Jensen, D. Lin, and B. C. Ooi. Query and update efficient b+tree-based indexing of moving objects. In *Proceedings of Very Large Databases (VLDB)*, pages 768–779, 2004.

[17] J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar. Provenance trails in the wings-pegasus system. *Concurr. Comput. : Pract. Exper.*, 20(5):587–597, 2008.

[18] D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pages 423–430, 2003.

[19] D. Lee, J. Choi, J. H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transactions on Computers*, 50(12):1352–1361, 2001.

[20] S. Majithia, M. S. Shields, I. J. Taylor, and I. Wang. Triana: A Graphical Web Service Composition and Execution Toolkit. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 514–524. IEEE Computer Society, 2004.

[21] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13:124–141, 2001.

[22] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[23] J. Qin and T. Fahringer. A novel domain oriented approach for scientific grid workflow composition. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[24] H. Samet. The quadtree and related hierarchical structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

[25] Y. L. Simmhan, B. Plale, and D. Gannon. Karma2: Provenance management for data-driven workflows. *International Journal of Web Service Research*, 5(2):1–22, 2008.

[26] H. Stockinger, A. Samar, K. Holtman, B. Allcock, I. Foster, and B. Tierney. File and object replication in data grids. *10th International Symposium on High Performance Distributed Computing (HPDC 2001)*, 2001.

[27] I. Taylor, A. Harrison, C. Mastroianni, and M. Shields. Cache for workflows. In *WORKS '07: Proceedings of the 2nd workshop on Workflows in support of large-scale science*, pages 13–20, New York, NY, USA, 2007. ACM.