K-nearest Neighbor Rule: A Replica Selection Approach in Grid Environment

Mohammad Rashedur Rahman
Department of Computer Science
University of Calgary
2500 University Drive, N.W
Calgary, Alberta, Canada T2N 1N4
{rahmanm}@cpsc.ucalgary.ca

Abstract

Grid technology is developed to share data across many organizations in different geographical locations. Data replication is a good technique that helps to move data because it caches data closer to users. The idea of replication is to store copies in different locations so it can be easily recovered if one copy at one location is lost. Moreover, if data can be kept closer to user via replication, data access performance can be improved dramatically. When different sites hold replicas, there are significant benefits realized when selecting the best replica. Network performance plays a major role in selecting a replica. However, current research shows that other factors such as disk I/O also plays an important role in file transfer. In this paper, we describe a new optimization technique that considers both disk throughput and network latencies when selecting the best replica. Previous history of data transfer can help in predicting the best site that can hold replica. The k-nearest neighbor rule is one such predictive technique. In this technique, when a new request arrives for best replica, it looks at all previous data to find a subset of previous file requests that are similar to it and uses them to predict the best site that can hold replica. In this work, we implement and test the k-nearest algorithm for various file access patterns and compare results with the traditional replica catalog based model. The results demonstrate that our model outperforms the traditional model for sequential and unitary random file access requests.

1. Introduction

The research community has recently recognized that advances in network bandwidth, processor speed and storage technologies make truly global sharing of distributed resources feasible. Grid computing evolves from the concept of integrating a collection of distributed computing resources to offer performance which are unattainable by any single machine. Grid computing allows coordinated resource sharing and problem solving collaboration in dynamic, multi-institutional and other potentially large-scale settings [6]. In a large amount of scientific disciplines such as global climate change, high energy physics and computational genomics, large data collections are emerging as important community resources. These large data stores must be able to be shared with researchers around the world. High performance Grid architecture allows us to identify the requirements and components common in different system and hence apply different technologies in a coordinated fashion to a range of data intensive petabyte scale application designed in Grid environment. As Grid technology is developed to share data across many organizations in different geographical locations, data replication proves to be a good technique that helps to move data by caching it closer to users [20]. The general idea of replication is to store copies of data in different locations so that data can be easily recovered if one copy at one location is lost. Moreover, if data

can be kept closer to user via replication, data access performance can be improved dramatically. Replication is a solution to many grid-based applications such as Climate data analysis and the Grid Physics Network [24] which requires responsive navigation and manipulation of large-scale datasets.

If multiple replicas exist, a replica management service is required. Replica management service discovers the available replicas and selects the best replica that matches the user's quality of service requirements. The replica selection problem can be divided into two sub-problems: 1) discovering the physical location(s) of a file given a logical file name, and 2) selecting the best replica from a set based on some selection criteria.

Since different sites hold replicas of a particular file, there is a significant benefit in selecting the best replica among them. Network performance plays a major role when selecting a replica. A slow network limits the efficiency of data transfer regardless of client and server implementation. One optimization technique to get the best replica from different physical locations is by examining the available bandwidth between the requesting computing element and various storage elements that hold replicas. The best site will be the one that has the minimum transfer time to transport the replica to requested site. Although, network bandwidth plays a major role in selecting the best replica, other source such as disk I/O plays an important role as well. Vazkhudai *et al*. [29, 30] show that disk I/O can account up for 30% of the transfer time in Grid environment. It is also noteworthy that disk capacity is increasing at the rate of two magnitudes per decade but the disk throughput is not increasing correspondingly. It has been shown that the ratio between disk capacity and disk throughput is increasing only at a single magnitude rate [17]. On the other hand, network bandwidth will increase three times per year. As the network latency drops significantly per year, disk IO will play a major role in the file transfer cost between different sites. Therefore, in this work, to measure Grid file transfer time, we extend the optimization technique by considering disk throughput with network bandwidth when selecting the best replica. The optimization function then returns the physical file name that will optimize the current network bandwidth between the requesting site and storage elements while considering the disk throughput of the storage elements.

In Grid environment, a replica catalog allows users to register files using a logical filename(s) or collection(s). It also provides mappings from logical names of files to the storage system location of one or more replicas of these objects. Generally the replica location service is centralized. Therefore, the requesting site must wait while all requests in the replica location queue are processed. This may cause substantial delays if there are a lot of pending requests. Further, as many sites hold the replica, the requesting site must find the best one by probing the network link and current state of disk storage. We can save this time, if we use the previous history of file transfers. Previous history sometimes provides some useful information such as the access frequency of a file, the access pattern between different file requests. If the requested file or its nearby file has been accessed recently, say from storage site x, we know that it will be accessed from site x again with a significant probability. Therefore, rather than getting information from the replica location service every time, the computing element can directly request the file from the previous requested storage site. To select the best replica locally, we use a simple technique called the K-Nearest Neighbor (KNN) rules [1]. KNN is extremely simple to implement and it is a good choice as no residual classifier needs to be built ahead of time. The traditional k-nearest classifier finds the k

nearest neighbors based on some distance metric by finding the distance of the target data point from the training dataset and then identifying the class from those nearest neighbors by some voting mechanism. The general strategy for KNN is to classify a new point as the majority of the k-nearest points have been classified.

The reminder of the paper is organized as follows. We give a brief introduction of Grid Computing and Globus Toolkit in Section 2. We present the related work in Section 3. The k-nearest rule and architecture of k-nearest replica selection are discussed in Section 4. The simulation setup is described in Section 5. An evaluation and comparison of our technique with the tradition replica catalog model is presented in Section 6. In Section 7, we propose a possible implementation of our algorithm in Globus Toolkit *version* 3. Finally in Section 8, we conclude and give directions for future work.

2. Grid Computing and Globus Toolkit

Over the last few years, "the Grid" in the context of resource sharing in distributed environments has gained a lot of attentions from the academic, government and commercial researchers. The term "Grid" refers to systems and applications that integrate and manage resources and services distributed across multiple control domains [11]. The Grid can accommodate very diverse resource types including storage devices, software, databases, objects, CPU power, files and others across many organizations in different geographical locations. Besides Grid computing aims to provide control of resource sharing and problem-solving collaboration with flexibility and security. The flexibility allows dynamic membership of a Grid in which Grid components can join and leave at will. Grid computing provides services with very large scale datasets and resource sharing in a global scale with heterogeneous systems.

The Globus Toolkit [12, 13] is a community based, open-architecture, open-source set of services and software libraries that provide the basic building blocks of grid-based applications and infrastructures. The toolkit addresses issues of security, information discovery, resource management, data management etc. The Globus [23] is generally recognized as the *de facto* standard in Grid computing. The major components of the toolkit include (1) The Grid Security Infrastructure (GSI), (2) Resource Management Services, (3) Information Services and (4) Data Transfer Protocol, *e.g.*, GridFTP.

Grid Security Infrastructure (GSI) [14] is the portion of the Globus Toolkit that provides the fundamental security services needed to support Grids. The public-key-based GSI provides single sign-on authentication and communication protection. The single-sign-on authentication makes it possible to authenticate a user just only once and thus create a proxy credential that a program can use to authenticate with any remote service on the user's behalf. GSI handles the mapping of the proxy certificate to the diverse local credentials and authentication/authorization mechanisms that apply at each site. GSI uses X.509 certificates [28], a widely used for PKI certificates as the basis of user authentication. GSI defines and X.509 proxy certificate to leverage X.509 for support of single sign-on and delegation. An X.509 certificate, in conjunction with an associated private key forms a unique credential set that a Grid entity uses to authenticate itself to other Grid entities. Each GSI certificate is issued by a trusted party known as a Certification Authority (CA). The CA is generally run by a large organization or commercial company.

Globus Toolkit's Meta Computing Directory Service (MDS) [9] provides a framework for discovering and accessing information about the structure and state of Grid resources. A resource obtains the information about other resources through GRIP (Grid Resource Information Protocol). GRIP supports both discovery and enquiry. Discovery is supported by searching where as enquiry corresponds to a direct lookup of information. For example, a service provider may provide various resources to other Grid entities. A broker (or user) can make a search on provider to obtain a set of results that roughly match a given criteria. After getting the search result, the broker can query the provider about the resource description and status. Besides, it can also make a subscription request to the provider such that resource update information will be delivered to it in a periodic fashion. A resource uses Grid Resource Registration Protocol (GRRP) to notify other entities that it is a part of Grid. GRRP is a soft-state [26] protocol. Soft-state is information that times out and must be periodically refreshed. Thus, using a soft state protocol is advantageous in that stale information does not need to be removed explicitly since it would time out eventually. Besides, it is resilient to failure because a single lost message does not cost irretrievable harm.

Grid Resource Allocation and Management (GRAM) [10] protocol provides reliable, secure remote creation and submission of a computational request to a remote computational resource and subsequent monitoring and control of resulting computation. The GRAM API provides functions for submitting and canceling a job or computational request. When a job is submitted a job handle is returned to the user. The user can use this handle later for monitoring and querying the status of the job. There are three major components in GRAM: gatekeeper, job manager and GRAM Reporter. The gatekeeper performs the authentication between remote user and resource and determines a local user name for the remote user. GSI mechanisms are used for authentication and credential delegation purpose. The gatekeeper then starts the job manager which actually handles the request. The job manager is responsible for starting the job on the resources and monitoring those while executing. The GRAM Reporter is responsible to publish the information about the scheduler and state into MDS. The state includes various information such as total number of nodes, currently active nodes, queue wait time and so on. A two-phase commit protocol is used for reliable invocation based on techniques used in the Condor system [12].

GridFTP is a data transfer protocol which is an extended version of FTP to provide secure, reliable and effective data transport of Grid data [2]. GridFTP supports GSI and Kereberos authentication with user controlled setting of various levels of data integrity and confidentiality. It also makes a third party to initiate, monitor and control file transfer between two other sites. It supports parallel data transfer through FTP command extensions and data channel extension. Moreover GridFTP can initiate stripped data transfer as well as it supports partial file transfer between two sites.

3. Related Work

Recently there has been a rise of interest in studying and modeling Grid environments. GridSim [4] is a Java-based discrete event simulator that supports modeling and simulation of heterogeneous Grid resources, users and application models. Nimord-G [5] is a resource broker that performs scheduling of parameter sweep task-farming applications on geographically distributed models. It supports deadline and budget based scheduling driven by market-based economic models. GridSim

simulates a Nirmod-G like Grid resource broker. It evaluates the performance of deadline and budget constrained cost and minimization scheduling algorithms. Though GridSim supports scheduling of jobs, it does not model anything related to data replication such as automated replica creation and/or selection.

Kavitha *et al.* [19] proposed a strategy for creating replicas automatically in a generic decentralized peer-to-peer network. The goal of their model is to maintain replica availability with some probabilistic measure. Their decentralized model has some advantages: there is no point of failure and it does not rely on a central monitoring scheme. The disadvantages are: nodes take decisions on partial information and sometimes it may lead to unnecessary replication.

Foster and Ranganathan [20] discuss various replication strategies. All these replication strategies are tested on hierarchical Grid architecture. The replication algorithms proposed are based on the assumption that popular files of one site will be popular at another site. The client site will count the number of hops for each site that hold replicas. According to the model, the best replica will be one that is the least number of hops from the requesting client. Though this is a simple heuristic, it does not consider current network bandwidth. Further, they assume that during file transfer between two sites, the link is busy and unable to transfer any other file through that link until the transfer ends. We do not constrain our model in this way so we select the best site by optimizing current network bandwidth and disk throughput Therefore, our model simulates the real Grid file transfer as well as allows other sites to transfer files through the same link at the same time but with reduced performance. Our replication algorithm is used by Grid sites when they need data locally and are based on both network attributes and data popularity while considering spatial and temporal locality.

Chervenak *et al.* [7] characterize the requirements for a Replica Location Service (RLS) and describe a data Grid architectural framework, Giggle (GIGa-scale Global Location Engine), within which a wide range of RLSs can be defined. An RLS is composed of the Local Replica Catalog (RLC) and the Replica Location Index (RLI). The LRC maps logical identifiers to physical locations and vice versa. It periodically sends out information to other RLSs about its contents (mapping) via a soft-state propagation method. Collectively, the LRCs provide a complete and locally consistent record of global replicas. The RLI contains a set of pointers from logical identifier to LRC. The RLS uses the RLIs to find LRCs that contain requested replicas. RLI may cover a subset of LRCs or cover the entire set of LRCs.

Allcock *et al.* [2] developed a replica management service using the building blocks of Globus Toolkit described above. The Replica Management infrastructure includes Replica Catalog and Replica Management Services for managing multiple copies of shared data sets. The Replica Catalog allows users to register files with a logical filename(s) or logical collection(s). It also maintains a mapping between the logical filename to one or more physical locations. It allows users and applications to query the catalog to find all existing copies of a particular file or collection of files. The replica catalog was implemented as a Lightweight Directory Access Protocol (LDAP) [18] directory. The management service however does not implement the full replica management functionality and does not enforce any replication semantics. It also does not provide any replica optimization or any other capabilities that we foresee for the immediate future based on our reading of the literature

In this paper, we introduce a higher level service that provides full dynamic replication functionality and it can use basic replica management services implemented in Globus to automatically create new replicas. The server can select the best one among replicas based on network and storage system performance.

4. K-Nearest Rule in Replica Selection

We define the replica selection problem in this way: find the best replica that will minimize the transfer time. This problem may be considered as a classification problem. Classification is the process of finding a set of models or functions that describe and distinguish data classes or concepts for purpose of predicting the class of objects whose class labels are unknown. Consider each training sample has n attributes: A_1, A_2,….., A_{n-1}, C, where C is the class attribute. Now if a new data sample whose attribute values for A_1, A_2,….., A_{n-1} are known, while the class attribute is unknown, the classification model predicts the class label of the new data sample using the known values of attribute values A_1, A_2,……, A_{n-1}. When a new sample arrives, K-Nearest classifier finds the k neighbors nearest to the new sample in the training space based on suitable similarity or closeness metric. A common similarity function is based on Euclidian distance between two data samples. For example, by considering attributes other than class labels and if $P=<p_1,p_2, \ldots\ldots, p_{n-1}>$ and $Q=<q_1,q_2, \ldots\ldots, q_{n-1}>$ are two data samples, the Euclidian similarity function is $d(P,Q)= \sqrt{\sum_{i=1}^{n-1} (p_i - q_i)^2}$. After finding the k-nearest samples based on the distance metric, the popularity class label of those k sample data points can be assigned to the new sample as its class. If there is more than one popular class label, one of them can be selected randomly.

In our simulated Grid environment, each computing site maintains a history of transfer when a file has been requested and transported to that client. The history includes the timestamp when the file transfer ends, the file index of the logical file and the storage site from which the file has been transferred. Each tuple in the history now includes three attributes with the storage site as the class label. After sufficient number of file transfers, each client builds a history that can be used as training space for a new tuple. At this point, when a new file request arrives, rather than finding the information from the replica catalog it finds the best site from the previous history of file transfer. The client determines the popularity of the class label, *i.e.*, storage site for k sample data points based on the distance metric with respect to two attributes: timestamp and file index. We use a weighted distance metric $d(P,Q)= \sqrt{\sum_{i=1}^{n-1} w_i (p_i - q_i)^2}$, where w_i represents the weight assigned each attribute. As timestamp are recorded to the milliseconds, we set w_1 to a fractional value and w_2 to unit value. File requests contain temporal locality, therefore, file index in distance measure will definitely improve the classification rate. For example, let us consider two sample points $P1=<p_{11}, p_{12}, S1>$ $P2=<p_{21}, p_{22}, S2>$ and a new sample point $P3=<p_{31}, p_{32}, ?>$ whose class label is unknown. Suppose $p_{21} > p_{11}$, that is P2 tuple has a recent transfer log than P1, and P3 has the same file index as P1, *i.e.*, $p_{32}= p_{12}$ but differs from P2 by one index number, *i.e.*, $p_{32}= p_{22}+1$. Now as the file contains temporal locality, we can say that if a logical file with index p_{22} is stored at site S2, the other file nearby its index say $p_{22}+1$ is also stored at site S2. Moreover, recent history shows that S2 is the best site to find a replica by considering current network and disk information. Therefore, the

recent history and temporal locality votes for site S2 rather than S1 although S1 is the site from which the requesting file was transferred previously.

The execution of a computing element is illustrated in Figure 1. The flow shows that if a computing element has sufficient history, it finds the best site using the k-nearest rule, otherwise it contacts the replica catalog to find all file replicas. Initially, each computing element gets information from the replica catalog and picks the best one that optimizes the transfer time considering current network and disk state. When each site has enough transfer history, it requests the best site by using the KNN rule. If the site found by the k-nearest rule has been deleted by this time because of insufficient disk space or another file's popularity exceeds than the requesting one, the computing element contacts the replica catalog to find other available replicas for that file. It will pick the best one to optimize the transfer time.

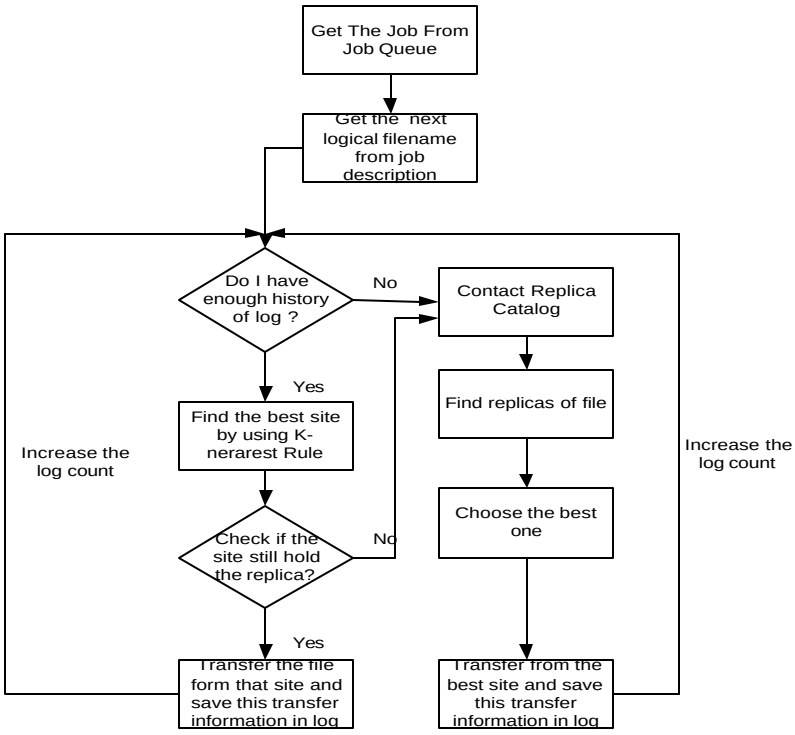


Figure 1: Execution Flows of a Computing Element.

5. Simulation

To evaluate our approach we use a simulation package called OptorSim [22], written in Java. OptorSim is a Data Grid simulator designed to allow experiments and evaluations of various replication optimization strategies in a Grid environment**.**

5.1 Architecture

The simulator is designed assuming that Grid consists of a number of sites, each consisting of zero or more computing elements and zero or more storage elements. The computing elements provide computational resource and the storage elements serve as data storage resources for submitted jobs. Sites without storage or computing elements act as routers. A resource broker acts as a meta-scheduler that controls job scheduling to different computing elements. A job in OptorSim must access a set of files which may be located at different storage sites. To get the physical locations of a logical file, each computing element consults with the replica management service. After getting information from the replica catalog, the replica management service returns the locations of the physical files for the requested logical file to the computing element.

In OptorSim, the optimization technique used to get the best replica from different physical locations is by examining the available bandwidth between the requesting computing element and various storage elements that hold replicas. The optimization technique does not consider the disk state when finding the best replica. The best replica will be the one that has the minimum transfer time with respect to current network state. We extend the optimization technique by considering disk throughput along with network bandwidth to find the best replica because file transfer time includes both network time and disk I/O time.

5.2 Grid Internals

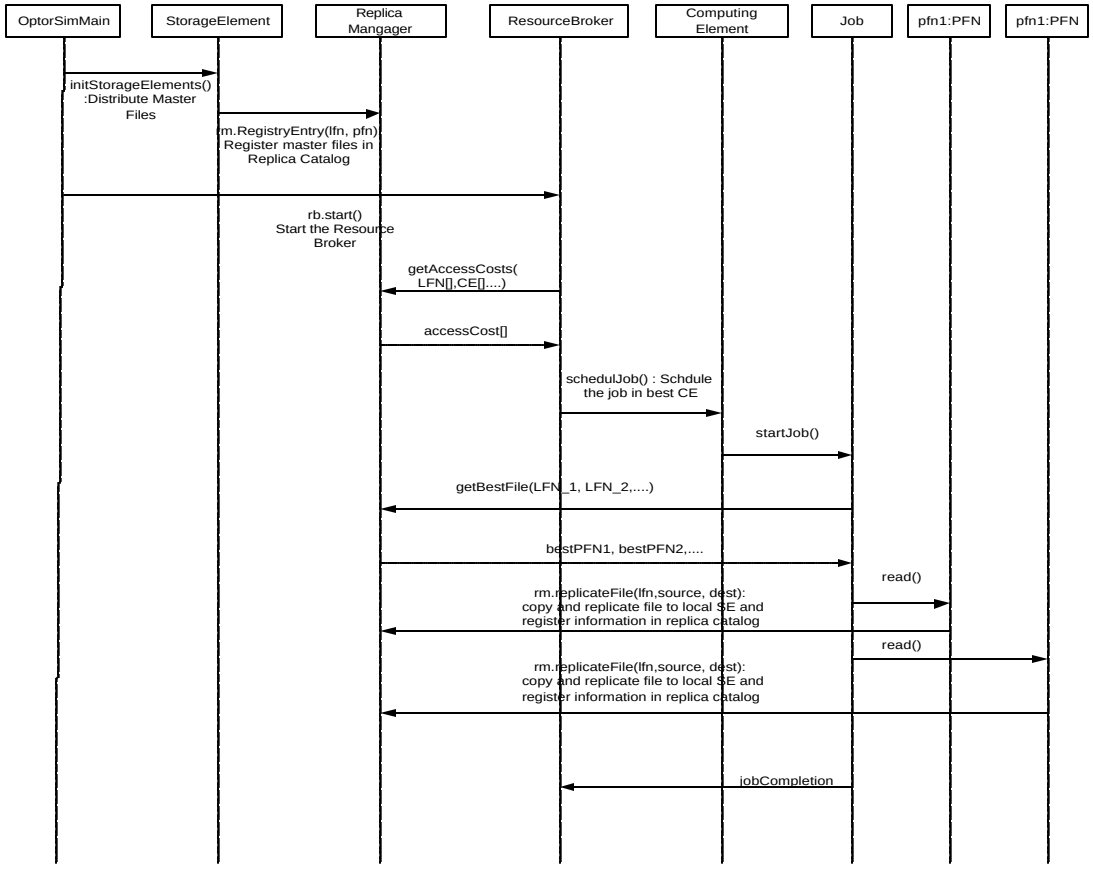


Figure 2: Typical interaction occurring in the Optorsim

Figure 2 depicts a typical interaction between various components in OptorSim. OptorSimMain is the entry point of the simulator.

- $t = t_1$: OptorSimMain initializes the storage elements and distributes the master files among different storage sites as specified in the configuration file.
- $t = t_2$: Storage elements register those master files in the replica catalog.
- $t = t_3$: OptorSimMain starts the resource broker.
- $t = t_4$: The resource broker calls getAccessCosts([LFN1..LFNN], CE[], …) to get the minimum access costs for all candidate CEs with respect to physical locations of the data. The information is used to determine the best CE for job execution.
- $t = t_5$: The resource broker schedules the job to the best client.
- $t = t_6$: The job starts.
- $t = t_7$: The job calls getBestFile([LFN1, LFN2], …) to determine the best physical locations of logical files considering current network and disk state.
- $t = t_8 ... t_{11}$: While reading the files from other storage sites, the CE makes a replica of each file to its own storage and registers this information into replica catalog.

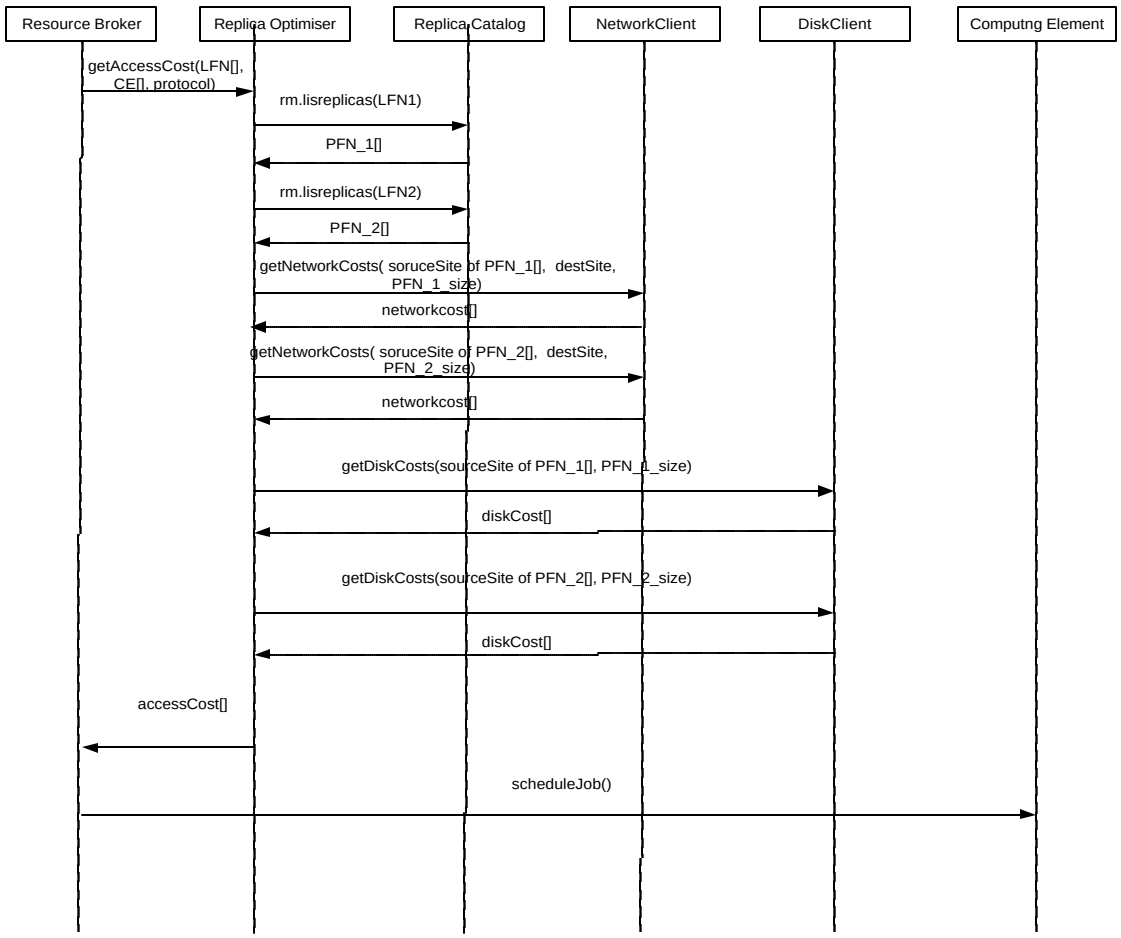


Figure 3: Internal behavior of the Replica Manager

Figure 3 shows the sequence of events that happen when the resource broker calls the function getAccessCosts().

- $t = t_{4,1}$: Replica Manager's listreplica() method is called to find the physical locations of the logical files. A number of physical filenames (PFN) will be found for a logical filename (LFN)
- $t = t_{4,2}$: Network cost is estimated for each site that holds a PFN corresponding to LFN.
- $t = t_{4,3}$: The disk cost will be estimated for getting PFN from storage site.
- $t = t_{4,4}$: The access cost for each CE are calculated based on the values returned by getNetworkCosts() and getDiskCosts().

5.3 Grid Configuration

The study of our optimization algorithm and K-Nearest replica selection algorithm was carried out using a model of the EU Data Grid Testbed 1 [3] sites and their associated network geometry. Site 8 is considered as CERN (European Organization for Nuclear Research). A master file contains the original copy of some data samples and can not be deleted.

Each circle in Figure 2 represents a Testbed site and a star represents a router. Each link between two sites shows the available network bandwidth. The network bandwidth is expressed in Mbits/sec (M) or Gbits/sec (G). The storage space and the disk throughput are also shown in braces. For example, Figure 2 shows that Site 0 has 80 GigaByte (GB) storage capacity with a disk throughput 260 M (Megabits/sec).

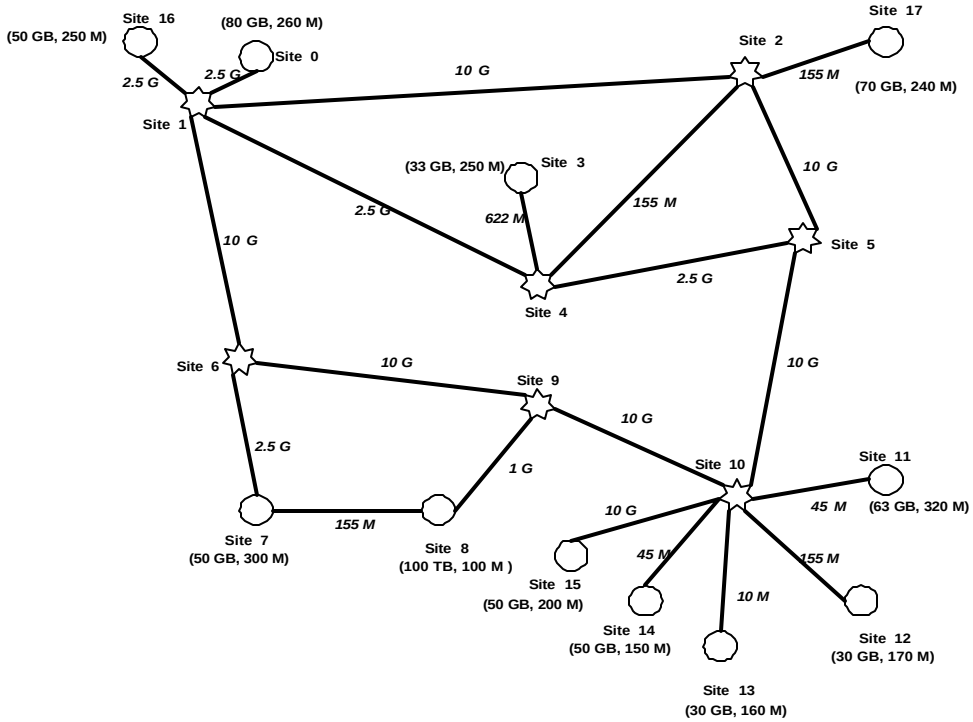


Figure 4: The EU Data Grid Testbed 1 sites and the approximate network and disk geometry.

5.4 Simulation Input

Our program gets input from three configuration files. One file describes the network topology, *i.e.,* the link between different sites, the available network bandwidth between sites, the size of disk storage of each site and the disk throughput of the disk storage. The second configuration file contains information about simulated jobs and the name of logical files which the job needs to access while executing. It also contains the index of each logical file and the schedule table for each computing element, *i.e.,* which jobs each computing element will run. The third configuration file initializes different parameters for running the simulation. These parameters may include information such as total number of jobs to be run, file processing time, delay between each job submission, maximum queue size in each computing element, file access pattern, the optimization algorithm used and so on.

As mentioned above, a job will typically request a set of logical filename for data access. The order in which the files are requested is determined by access pattern. In this simulation we experimented with four access patterns: *sequential* (files are accessed in the order that has been stated in the job configuration file), *random* (files are accessed using flat random distribution), *unitary random* (file requests are one element away from previous file request but the direction will be random), *Gaussian random walk* (files are accessed using a Gaussian distribution). Those file access patterns are shown in Figure 5.

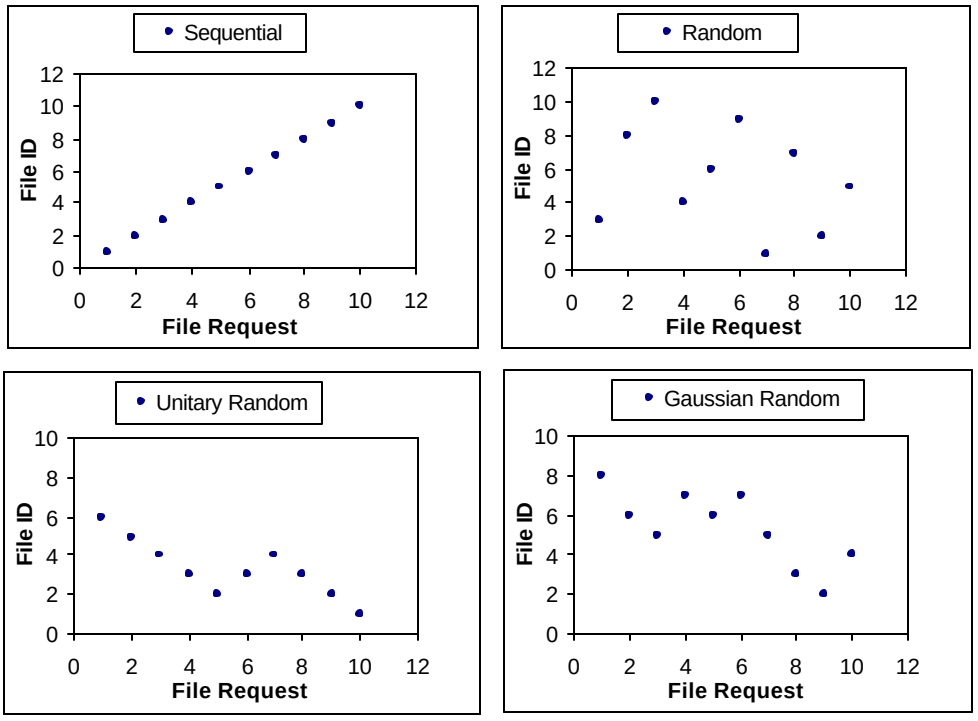


Figure 5: Various File Access Patterns

6. Simulation Results

Our simulations are run for 200 jobs of various types. Each job requests file of different sizes. We experimented with different file transfers starting from 10 MB (Mega Bytes) up to 1 GB (1 Giga Bytes). The jobs are submitted with a fixed probability such that some jobs are more popular than

others. Each job is submitted at 25 milliseconds interval. Time taken for completing a job is equivalent to the waiting time in the queue in the computing element and the execution time at computing element. Time for probing network bandwidth and disk state is considered 10 milliseconds. Our scheduling algorithm takes into account both locality of files requested by jobs and computing element loads. The calculation of the file accessing cost is implemented by the getAccessCosts() method described in the previous section. For each computing element that is candidate for scheduling, the resource broker calls the getAccessCosts() method with a list of files required for the job. The function consults with the replica catalog to find all the replicas of each file, then calculates the time to access each replica from the give computing element by examining the current network and disk state. By summing the times to access cost to access the best replica of each file, getAccessCosts() returns the estimated file access time the job would have if scheduled to the computing element. The scheduling algorithm schedules a job to the computing element that has the minimum access cost. We use a simple replication approach called LRU (Least Recently Used) for replica creation. In this technique, each computing element stores a replica of the file that it has requested into its local storage element. If the storage element has sufficient space the file will be stored there. Otherwise the storage element makes space for the newest file by deleting the least recently used file(s). Initially each site builds a history of 50 initial file transfers. We choose 50 as an arbitrary value to see how it effects on the performance of our algorithm. In the KNN approach each computing element finds the best one by the k-nearest rule rather than requesting the replica catalog. For the sequential access pattern, there is a linear dependence of file access patterns. So, it will be more intuitive to use a small value of k for sequential access pattern because large value of k will cause other non nearby neighbors to vote and decreases the performance of the algorithm. The remaining access patterns contain randomness so we should use a value that is neither very large nor small. Therefore, we choose the value of k=2 for sequential access pattern and k=5 for other access patterns. We also vary the value of k for some initial run in our simulation and we get the highest accuracy when we take k=2 for sequential pattern and k=5 for other access patterns. Recall that each site will be running a varying number of jobs such that some site may run only a few jobs that result in only a few file transfers. In our experiment, every site except Site 13 and Site 15 are loaded with jobs which request more than 50 transfers. Therefore, we will not include Site 13 and Site 15 in our analysis.

Figure 6 documents the performance of our KNN algorithm and the traditional replica catalog model with respect to total job times. In this case all master files are in CERN (Site 8). For the sequential and Unitary Random access pattern, our enhanced algorithm outperforms the traditional model. As the file will be accessed in the order specified in the job configuration file for sequential requests, *i.e.* file access patterns are sequential, the KNN rule will find a recent transfer history of neighbor(s) of a requested file. The same argument can be made for Unitary Random access pattern because new file requests will be one element away from previous request but the direction will be random. However, we do not see much improvement in the Uniform and Gaussian Random file access pattern. This is because for those patterns files are accessed without any relation of previous requests and the same file can be requested multiple times leaving some files unaccessed. Therefore, though from previous history, KNN votes for a site x that can hold a replica of the newly requested file y: site x may never transfer y from CERN (Site 8) so it does not hold any replica for that file. However, this will not happen for sequential file access patterns because a job will request files one after another and the site that runs that job creates replicas of all files needed to run the job.

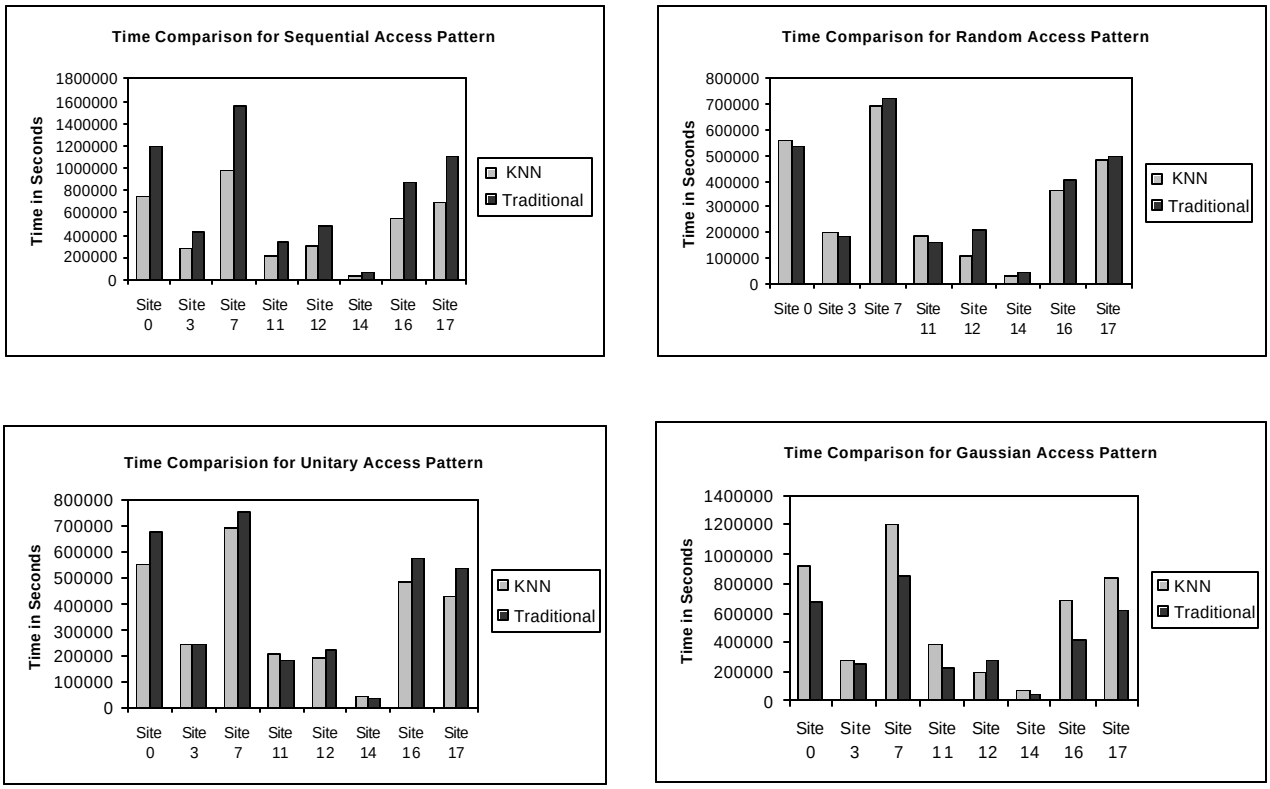


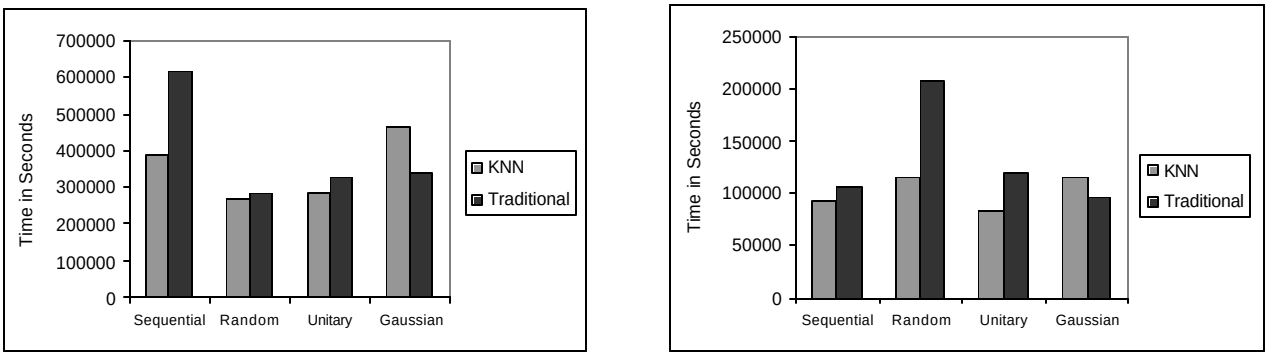
Figure 6: Comparison of Total Job Times for K-Nearest Rule and Traditional Model

Sequential	Site	Match	Mismatch	**Random**	Site	Match	Mismatch
	Site 0	314	3		Site 0	281	36
	Site 3	100	6		Site 3	93	13
	Site 7	1508	3		Site 7	1471	40
	Site 11	133	0		Site 11	132	1
	Site 12	50	6		Site 12	47	9
	Site 14	6	2		Site 14	7	1
	Site 16	191	2		Site 16	186	7
	Site 17	238	0		Site 17	234	4
Unitary	Site	Match	Mismatch	**Gaussian**	Site	Match	Mismatch
	Site 0	286	31		Site 0	275	42
	Site 3	96	10		Site 3	93	13
	Site 7	1461	50		Site 7	1462	49
	Site 11	131	2		Site 11	132	1
	Site 12	50	6		Site 12	47	9
	Site 14	6	2		Site 14	6	2
	Site 16	168	25		Site 16	182	11
	Site 17	231	7		Site 17	226	12

Table 1: Classification and Misclassification Rate for Different File Access Pattern

Table 1 presents the classification accuracy of the KNN rule for various file access patterns when all files are distributed in the CERN. If the KNN rule returns the same site as returned by the traditional replica model, the file transfer is classified as matched; otherwise it is classified as mismatched. We see that for sequential access pattern the accuracy is highest whereas for the Gaussian Random access pattern the accuracy is the lowest. This result also complies with Figure 6. There is a strong correlation between accuracy and job throughput as one would expect intuitively.

Table 1 and Figure 6 contain some discrepancies. For example, we see that for Site 16, KNN misclassifies 11 and 25 file transfers in Gaussian access pattern and Unitary Random access pattern, respectively. However, Figure 6 shows a better throughput in Unitary Random access pattern than Gaussian access pattern. This is because in the Gaussian access pattern, KNN makes a misclassification for a large file transfer, *e.g.*, 1 GB whereas in Unitary access pattern it mismatches with the traditional model for only small file transfers *e.g.*, 10 MB transfers. The misclassification for large file transfer costs more than a couple of small file transfer misclassifications.



(a) (b)

Figure 7: a) Mean job time when CERN holds master files b) Mean job time when different sites hold master files.

Figure 7 shows the comparison of mean job time between two methods, when master files are distributed randomly and when master files are distributed to the Site 8 (CERN). If CERN holds the master files alone, all other sites have to transfer files from CERN and it generates a huge amount of network traffic at CERN. Moreover, each site has to transport file from CERN's storage element, therefore, the disk storage at CERN will be busy for disk I/O and it will increase the response time. However, if the master files are distributed to different sites, a site can transfer files from various locations rather than from a single location. As a result it reduces both network traffic and disk I/O time. Therefore, we get a better mean job time when master files are distributed randomly. However, in both cases, we see that our KNN algorithm has better mean job time compared to traditional model for various access patterns except Gaussian pattern.

Table 2 represents a comparison of accuracy of our KNN algorithm with respect to traditional model for different distribution methods of master files. The overall accuracy of our KNN algorithm is 97% when master files are distributed in random sites and we get 96% accuracy when distributed in CERN alone.

Sequential	Master File Distribution	Match	Mismatch	**Random**	Master File Distribution	Match	Mismatch
	Random	2539	23		Random	2368	194
	CERN	2540	22		CERN	2451	111
Unitary	Master File Distribution	Match	Mismatch	**Gaussian**	Master File Distribution	Match	Mismatch
	Random	2524	38		Random	2468	94
	CERN	2429	133		CERN	2423	139

Table 2: Comparison of Accuracy

7. Possible Implementation in GT3

Globus Toolkit *version 3* (GT3) is the implementation of the Open Grid Services Architecture (OGSA) [15], an initiative that is recasting Grid concepts within a service oriented framework based on Web Services. Web Service is a platform and implementation independent software component that can be discovered, combined, and recombined to provide a solution to the user's problem. The Java and XML are the prominent technologies for Web Services. Web service has several advantages: service providers can publish detailed description of their functionalities through web services, services can be easily discovered through a standard mechanism, it supports different protocol bindings such as HTTP and SOAP and web service can be easily incorporated into or composed with other services. However, web service has disadvantages too: it is stateless and non-transient. Grid service, an extension of web service solves the problems of Web Services by a factory approach. Grid services also provide some necessary tools such as call back functions to effectively manage lifecycle and service data elements for describing its service. Besides, Grid services support notification policy to notify the changes to subscribed clients. OGSA defines a common and standard architecture for computational and storage resources, network programs and databases. It defines the Grid services which are the center of this architecture. Open Grid Service Infrastructure (OGSI) [15] is a formal and technical specifications of the concepts described in OGSA including Grid services. GT3 is a software tool that allows us to program in Grid based applications.

Figure 8 depicts a possible implementation of our algorithm in GT3. We propose Condor-G [16] as resource broker or meta-scheduler of our algorithm. A user will submit his request for running jobs to Condor-G scheduler. The request will include a list of jobs. Each job is described by a set of files that are required to run that job. The Condor-G scheduler creates a new GridManager daemon to submit and manage those jobs. The GridManager process will handle all jobs for a single user and terminate when all jobs will be finished. The GridManager process requires the functionality of the Replica Management Service, in particular the optimization functions in order to determine the best computing element on which to schedule a job. Replica Management Service (RMS) factory provides such service to GridManagger in our algorithm. RMS factory creates a RMS instance that will contact the Replica Catalog to find the physical locations of logical files required to run the job. It will then ask the Replica Selection Service to find the best site for hosting the current job by considering current network and disk state. Network Weather Service (NWS) [32] will be used to sense network bandwidth and IOSTAT [27] for disk state.

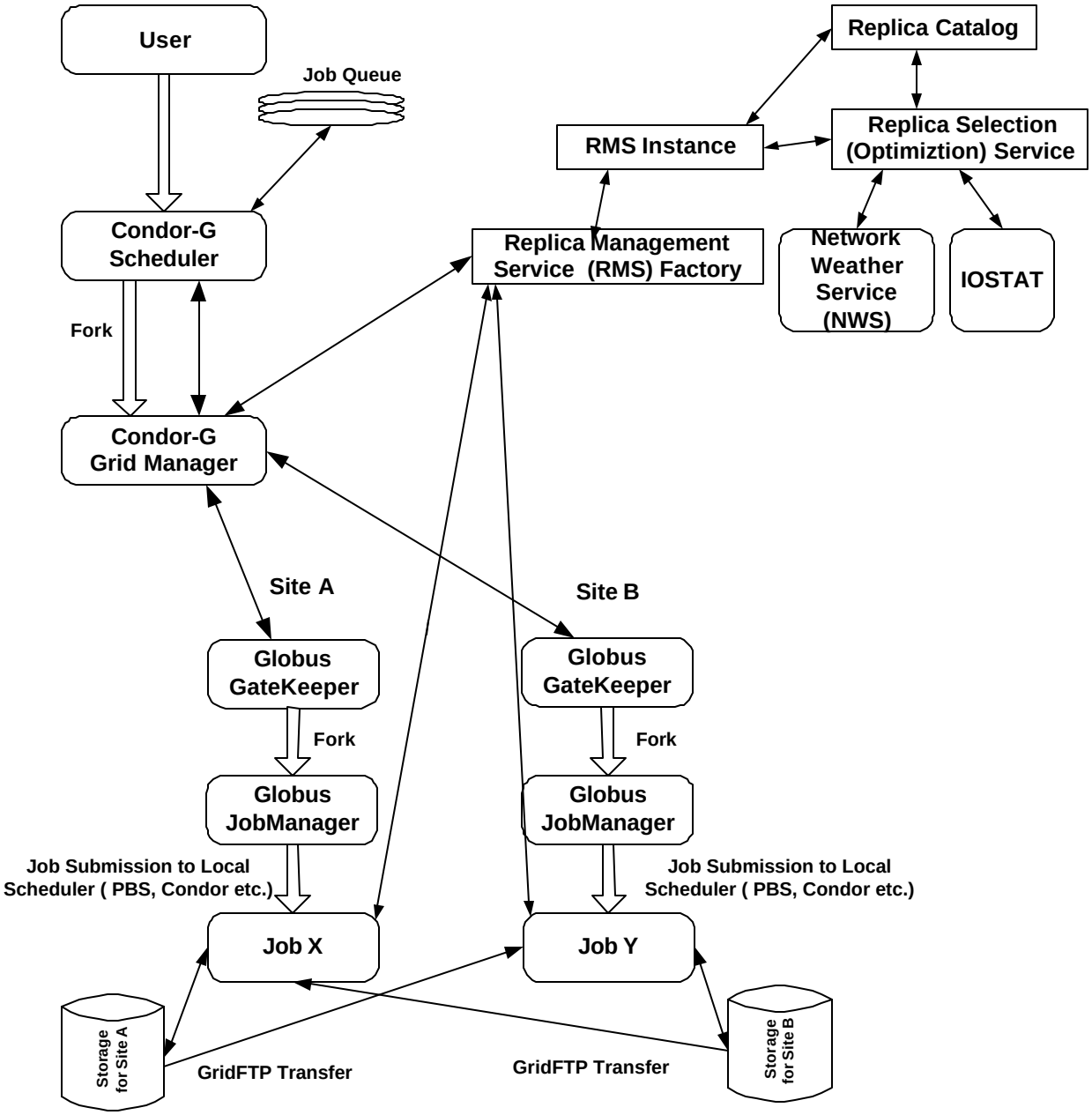


Figure 8: A possible implementation of our algorithm in GT3

Getting the result from RMS factory, the GridManager hosts the job to the best site through GlobusGateKeeper. Next, the Globus JobManager submits the job to the execution site's local scheduling system. It is mentioned before that a set of files are required to run a job. The information about physical locations of logical files can be found from Replica Catalog. Therefore, each site will contact the RMS factory to find the physical locations of the files that are needed to run the job. If the files are in current site, the job will access files from local storage. Otherwise, it will find the best site hosting replica(s) by querying the Replica Selection Service in the same way as described above. Next, the file(s) will be transferred from the best site by GridFTP. Each site creates a new replica of the file that it transfers from other site into local storage. Then each site

registers the new replica in Replica Catalog. Besides, each site updates its transfer log reflecting the current file transfer. The transfer log will be used as training space of our KNN algorithm when each site has a sufficient history of file transfers. Updates on job status are sent by the JobManager back to the GridManger, which then updates the Scheduler where the job status will be stored persistently.

A GSI proxy credential is used by the Condor-G agent to authenticate with remote resources on the user's behalf. To reduce the user hassle in dealing with expired credentials, Condor-G could be enhanced with a system like MyProxy [25]. The information about Replica Management Service can be found from MDS. We have a plan to develop Replica Management Service and Replica Selection Service (RSS) as Grid Service. We will use the Globus Toolkit's Replica Catalog [2] component for indexing logical file to physical locations and vice versa. The GridManager or a computing site connects to the RMS factory by Grid Service Handle (GSH) and logical filename (s). The RMS and RSS grid service interface will be written in WSDL (Web Service Description Language) [8]. It can be also generated from a Java interface. For Grid service implementation, java classes will be used. These Java classes provide implementation of methods that are specified in the service interface. The deployment descriptor will be written in WSDD (Web Service Deployment Descriptor). WSDD tells the web server how it should publish the RMS and RSS service.

8. Conclusion and Future Work

In this paper, we presented a model that utilizes a simple data-mining approach called the k-nearest rule to select the best replica from a number of sites that hold replicas. To select the best replica we designed an optimization technique that considers network latency and disk state. We studied different file access patterns on the performance of the k-nearest model and compared our results with traditional algorithm. Our k-nearest rule shows a significant performance improvement for sequential and unitary random file access pattern.

As part of future work, we plan to design an adaptive k-nearest neighbor algorithm so that it can switch to traditional model when it encounters misclassification with traditional model for consecutive file transfers. Further, rather than using the fixed value of k, we can use an adaptive value of k that will change dynamically as the simulation runs to improve performance dynamically. Moreover, before making a decision about the best site for large file transfers each site can verify the best one by varying the value of k. We also have a plan to implement our algorithm in WestGrid [31] middleware to see its performance in real Grid Environment.

References:

[1] Aha, D., D. Kibler, and M. Albe. *Instance based learning algorithms*. Machine Learning, 6:37--66, 1991.

[2] Allcock, B., J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke. *Secure, Efficient Data Transport and Replica Management for High Performance Data-Intensive Computing*. IEEE Mass Storage Conference, 2001

[3] Bell, Willam., D. G. Cameron, L.Capozza, A., P. Millar, K. Stockinger, and F. Zini. *OptorSim - A Grid*

Simulator for Studying Dynamic Data Replication Strategies. International Journal of High Performance Computing Applications, 17(4), 2003.

[4] Buyya, R., and M. Murshed, *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing,* The Journal of concurrency and Computation: Practice and Experience (CCPE), Volume 14, Issue 13-15, Wiley Press, Nov.-Dec., 2002.

[5] Buyya, R., D. Abramson, and J. Giddy. *Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid*, HPC Asia 2000, May 14-17, 2000, pp 283 289, Beijing, China.

[6] Chervenak, A., I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. *The Data Grid: Towards and Architecture for the Distributed Management and Analysis of Large Scientific Data Sets.* Journal of Network and Computer Applications, 23(3):187-200, 2000.

[7] Chervenak, A., E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B, Schwartzkopf, H, Stockinger, K. Stockinger, B. Tierney. *Giggle: A Framework for Constructing Sclable Replica Location Services.* Proceedings of Supercomputing 2002 (SC2002), November 2002

[8] Christensen, E. Web *Services Description Language (WSDL) 1.1*. W3C Note, 15 Mar. 2001

[9] Czajkowski, K., S. Fitzgerald, I. Foster, C. Kesselman. *Grid Information Services for Distributed Resource Sharing*. Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), *IEEE Press, August 2001*

[10] Czajkowski, K., I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. *A Resource Management Architecture for Metacomputing Systems*. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998

[11] Foster, I., C. Kesselman, and S. Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual organization.* International Journal of High Performance Computing Applications,15 (3), 2001.

[12] Foster, I., C. Kesselman. Globus: *A Metacomputing Infrastructure Toolkit.* Intl Journal of Supercomputer Applications, 11(2):115-128, 1997.

[13] Foster, I. and Kesselman, C. *Globus: A Toolkit-Based Grid Architecture.* In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 259-278.

[14] Foster, I., C. Kesselman, G.Tsudik, and S. Tuecke. *A Security Architecture for Computational Grids.* In ACM Conference on Computers and Security, 1998, 83-91.

[15] Foster, I., C. Kesselman, J. Nick, S. Tuecke. Grid *Services for Distributed System Integration.* Computer, 35(6), 2002.

[16] Frey, J., T. Tannenbaum, M. Livny, I. Foster, S. Tuecke. *Condor-G: A Computation Management Agent for Multi-Institutional Grids.* Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[17] Gray, J. and P. Shenoy. *Rules of Thumb in Data Engineering*. International Conference on Data Engineering ICDE2000, 2000, San Diego: IEEE Press.

[18] Howes T.A and M. Smith. *A scalable, deployable directory service framework for the internet.* Technical report, Center for Information Technology Integration, University of Michigan, 1995.

[19] Kavitha, R., A. Iamnitchi, and I. Foster. Improving *Data Availability through Dynamic Model Driven Replication in Large Peer-to-Peer Communities.* Proceedings of Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop, Berlin, Germany, May 2002.

[20] Kavitha, R., and I. Foster, *Design and Evaluation of Replication Strategies for a High performance Data Grid, i*n Computing and High Energy and Nuclear Physics 2001 (CHEP'01) Conference.

[21] Livny, M. *High-Throughput Resource Management.* In Foster, I. and Kesselman, C. eds. The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999, 311-337.

[22] OptorSim, *http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html*

[23] The Globus Project, *http://www.globus.org*

[24] The GriPhyN Project, *http://www.griphyn.org*

[25] Novotny, J., S. Tuecke, and V. Welch. *An Online Credential Repository for the Grid: MyProxy*, Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001

[26] Raman, S. and S. McCanne, *A Model, Analysis, and Protocol Framework for Soft State-based Communication.* Computer Communication Review, 1999. **2**9(4).

[27] *SYSSTAT Utilities Homepage*, *http://perso.wanadoo.fr/sebastien.godard/*,

[28] Tuecke, S., Engert, D., Foster, I., Thompson, M., Pearlman, L. and Kesselman, C. *Internet X.509 Public Key Infrastructure Proxy Certificate Profile.* IETF, Draft draft-ietf-pkix-proxy-01.txt, 2001.

[29] Vazhkudai, S., J. Schopf. *Using Disk Throughput Data in Predictions of End-to-End Grid Transfers.* Proceedings of the 3rd International Workshop on Grid Computing (GRID 2002), Baltimore, MD, November 2002.

[30] Vazhkudai, S., J. Schopf. *Using Regression Techniques to Predict Large Data Transfers.* The International Journal of High Performance Computing Applications (IJHPCA), special issue on Grid Computing: Infrastructure and Applications, August 2003.

[31] The WestGrid Project, *http://www.westgrid.ca*

[32] Wolski, R., *Dynamically Forecasting Network Performance Using the Network Weather Service.* Cluster Computing, 1998.