Dynamic Data Replication for Short Time-to-Completion in a Data Grid

Ralf Vamosi$^{1[0009-0002-8712-6564]}$ and Erich Schikuta$^{1[0000-0002-4126-4243]}$

University of Vienna, Faculty of Computer Science, Vienna, Austria
{ralf.vamosi,erich.schikuta}@univie.ac.at

Abstract. Science collaborations use computer grids to run expensive computational tasks on large data sets. Tasks as jobs across the network demand data and thereby workload management and data allocation to maintain the computational workflow. Data allocation includes data placement with different replication factors (multiplicity) of data.
The proposed data replication & allocation model can place multitudes of subsets of a data population in a distributed system, such as a computer cluster or computer grid. A stochastic simulation with a data and computing example from the ATLAS Physics Collaboration shows its potential usability in one of the largest Computing Grids. This paper showcases data allocation with different replica factors and various numbers of subsets to improve the overall situation in a computer network.

Keywords: data replication · data placement · data partitioning · data-intensive computing

1 Introduction

Data replication is essential for quicker response or time-to-completion in computer clusters and data grids. Replication involves generating copies of data on different nodes or systems in the network. Data parallelization reduces the time required to read and complete computations, balances the processing load, and improves system resilience. The base case has a replication factor of 1, meaning that only one copy of the data is stored in the system. This is often referred to as a single-replica or non-replicated setup. Other cases are replication factors of 2 or more. A fraction of an integer means that a data subset is replicated.

The motivating use case for the presented method is the ATLAS physics collaboration. The scientific collaboration stores data as files across a worldwide computing grid with 160 geographically distant sites. The data is organized into labeled data sets or file collections that can change over time, and the files are the primary units for computational jobs.

Delay or latency in computer networks, especially across wide area networks (WAN), can cause delays in workflow, and how data replication can improve the situation by promoting the local processing of data sets. However, jobs must be allocated to appropriate sites with the necessary resources to support the required computation. Load balancing ensures that jobs are placed at sites that

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36024-4_52

can support the necessary computations. The site with input data is preferred as the target to fully utilize processing resources and reduce delays.

In the same way, data sets cannot be randomly placed on sites or nodes due to storage limitations leading to two major points: Every time a job needs to run, it may transfer files as its input and thereby use a system bottleneck, the interlinking network (WAN) between sites. The network consists of non-uniform sites providing resources with considerable differences. Furthermore, network connections vary from site to site due to cost and provision limitations.

2 State of the Art

Data replication has a long history. The technique is referred to with multiple terms and seen from different points of view. The main reasons for replication are data backup and data parallelization. It was investigated when distributed databases were used, so parallelization was to be utilized. Data replication is NP-complete due to the combinatorial explosion in the selection process. Research on data replication presents various models and algorithms for file placement and replication in distributed systems. The models focus on factors such as storage capacity, network bandwidth constraints, load balance, reliability, and cost trade-offs. They also use different approaches, including deterministic and stochastic, and hybrid models that combine both. Many software architectures have been proposed for parallel systems [2]. Mathematical models like game theoretic and Markov decision process models also make file placement decisions.

The paper of [3] investigates the mathematical modeling of a network of nodes with replicated data files, focusing on transactions involving multiple files. The authors consider two types of transactions, namely, query and update traffic, and use a linear cost model to illustrate optimal file assignment to nodes. They present bounds on the number of file copies required in the network based on the query, update traffic, and derive a test to determine the optimal configuration.

An algorithm for dynamic replication of a data object in distributed database systems that adapts to changes in the read-write pattern, continuously moving the replication policy toward an optimal one, is presented in [7]. The algorithm can be integrated with concurrency control and recovery mechanisms and provide a lower bound on the performance of any dynamic replication algorithm.

The paper of [4] proposes replication management services and protocols for efficient and fast access to large and widely distributed data in Data Grids. The cost model for replication decisions consider factors such as run-time read/write statistics, response time, bandwidth, and replica size. The system evaluates the costs and performance gains of creating each replica and organizes them in hierarchical and flat topologies to minimize inter-replica communication costs.

The paper from [5] summarizes research on genetic algorithms (GA) to optimize data replication in large distributed systems and reduce network traffic delays. The authors compare their GA approach to a greedy method in a static scenario with constant read/write demands and also propose a hybrid GA approach to handle changing patterns. The experiments are conducted with up to 25 sites and 90 data objects.

The paper of [1] explores the energy efficiency and bandwidth consumption of data replication in cloud computing data centers, considering the QoS improvement achieved by reducing communication delays. A mathematical model and extensive simulations are used to evaluate the performance and the trade-offs.

The paper from [6] proposes an approach based on evolutionary clustering to improve data allocation in distributed systems. Popularity information allocates data items to storage nodes, including a stochastic search with a fast clustering method. The method improves data allocation without changing the number of replicas in the global file system while considering storage constraints and improving data access performance.

3 Optimization Method

The proposed heuristic optimization solves data replication and partitioning by inferring from a large data population into smaller subsets for replication and placement. Major factors in parallel computing are data available on multiple nodes, the different types of data, and the read frequencies. Read patterns introduce some degree of unpredictability in the job execution process among sites. The uncertain nature is modeled as probability mass functions (PMF).

Our model shows how patterns in data are used to fulfill replication for a distributed workflow. For an overall improvement, average values are taken from network connections. Dynamic replication is supported because runs can be repeated as often as necessary for any data share and replication factor. The method returns a tuple of subsets replication runs. The optimization process aims to minimize the target cost/loss by conditionally finding subsets in the data population:

$$\begin{aligned} &\underset{\boldsymbol{A}}{argmin}\ cost_{network}(\boldsymbol{A}) \\ &s.t. \\ &\boldsymbol{A}^T \times \boldsymbol{w}_{point} \leq \boldsymbol{w}_{site} \end{aligned} \tag{1}$$

, where the parameter $\boldsymbol{A}$ defines the objects or points per subset $A_1, ..., A_N$. $\boldsymbol{A}$ is a slim matrix with one row per data point and one column per subset. A '1' indicates one replica (row) in a subset (column). Multiples are possible for higher replication factors. The network cost $cost_{network}(\boldsymbol{A})$ will be introduced next. $\boldsymbol{w}_{point}$ is a column vector representing the sizes for data points, and $\boldsymbol{w}_{site}$ is a column vector holding the maximum sizes for the final N subsets.

During search steps, the method keeps the parameters if it results in improvement and repeats until no further satisfaction can be achieved. The effort to improve continuously increases as better data sampling becomes harder on a higher baseline. **Algorithm 1** outlines this process as pseudo-code: The function setSystemParameters() sets the values for system parameters and variables before each run, including network variables, site variables, datasets, and job sets. *setupClustering*() setups and selects the clustering method. Due to space limitations, we are unable to provide any discussion. Next-neighbor is the base method. *setSeeds*() memorizes the seed positions and partially alternates them. The cost function *calculateCost*() depends on its parameters, system variables, and meta parameters. Its parameters are the data set $\boldsymbol{X}$ and a tuple of data placement at each site or node denoted as $A = (A_1, ..., A_N)$. Some A_i may be empty. The method adds subsets $A' = A'_1, ..., A'_N$ for each replication to the allocated data $A = A_1, ..., A_N$. The final data allocation always depends on the replication case. An optimization yields the initial data placement A merged with A', where the tuples are joined element-wise (denoted by operator +), incorporating constraints for capacities. The free space is $w' \leftarrow w_i - |A_i|$ for the subset with index i depending on the maximum size w_i and the data A_i already placed at the i'th site or node. Depending on the cost, the update process continues and returns the dominating solution stored as A''.

Algorithm 1 Data sampling algorithm with clustering for data replication. Input parameters are data set X, subsets of data placement $A = A_1$ to A_N, maximum subset sizes $w = w_1$ to w_N, and the number of clusters k.

```
function GETSUBSETS(X, A, w, k)
    for $i$ in 1 to $N$ do
        $w' \leftarrow w_i - |A_i|$
        $k' \leftarrow$ GetNumberOfSeeds(w', i, k)
        setSeeds(X, $k'$)
        $A'_i \leftarrow getSubsetComplementary(X, A_i, w')$
    end for
    return $A' = A'_1, ..., A'_N$
end function
setSystemParameters()
$cost \leftarrow$ calculateCost($X$, $A$)
$counter \leftarrow 0$
while termination condition not met do
    setupClustering(counter)
    $A' \leftarrow getSubsets(X, A, w, k)$
    $cost' \leftarrow$ calculateCost($X$, $A + A'$)
    if $cost' < cost$ then
        $A'' \leftarrow A'$
        $cost \leftarrow cost'$
    else
        $counter \leftarrow counter + 1$
    end if
end while
return $A'', cost$
```

Within processes performing the algorithm, the stochastic search of subsets results better over time, and the best solution converges to the optimum. However, it will generally not reach the global optimum since of the large non-convex search space. A mechanism intended to prevent a process from getting stuck is to monitor and count the failures to improve the solution. If stuck in a local minimum, it cannot proceed, and the parameters are then randomized again to start over. The data sampling is based on clustering, whose parameters are type, weights, and number of clusters. Type is, e.g., soft or hard clustering. The parameters alternate over time. The cluster seeds may sometimes spread if the solution becomes better this way. An opposite case would be, for example, one cluster for a subset that stays as one cluster in the final solution.

In the context of data replication and data placement, the primary factor is the data transfer time, which will be selected as the target cost/loss. The $cost_{network}(\boldsymbol{A})$ penalizes external data transfers from site to site.

$$cost_{network}(\boldsymbol{A}) = \sum_{j \in \{jobs\}} cost_{network}(\boldsymbol{A}, j) \tag{2}$$

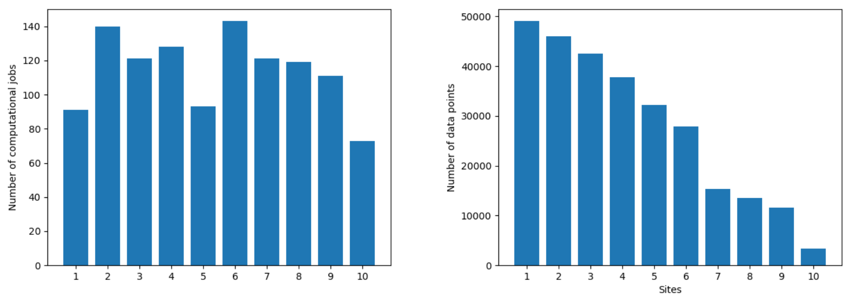
, where $\boldsymbol{A}$ is the allocation parameter as before. j is a job from the job batch (validation). $cost_{network}(\boldsymbol{A}, j)$ denotes an affine approximation dependent on the non-local files for job j given $\boldsymbol{A}$. The value per file depends on the file size and the bandwidth between the source and the target.

4 Justification and Evaluation

Resources, such as computing resources, storage, and network bandwidth, are randomly chosen for each simulation run. Random selection means for storage, for instance, that each node gets a random value within a range to store files. At the end of value sampling for parameters, the situation must be such that the sites or targets (subsets from the data perspective) provide room for all replicas. Additionally, there are several job types and several data types to represent the real-world scenario. This shall represent the essential types or classes if there are more than that. The simulation runs multiple times to average out uncertainties on parameters and variables. The results ensure a fair evaluation without picking random or biased values in the first place. The **model** of the parallel system consists of a network with sites, jobs as basic processing units, and data points representing datasets or collections of files:

Network connections of sites are each sampled before a simulation run relative to the maximal connection bandwidth:
$bandwidth_i \in [0.1, ..., 1]$ for all sites $1, ..., N$

Sites or nodes hold data and process jobs on the data. Each site has several different resource types, such as high-mem (high memory) or GPU. A site with small storage must rely more on others to provide files to be processed. The model mimics that each node handles different job types differently quickly. The number of jobs is expressed as a value in the probability mass functions depending on type and size, such as in Fig. 2.

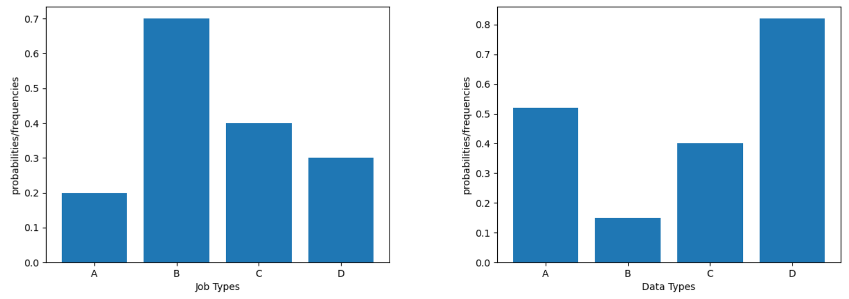


(a) Summed up running jobs per site. (b) Stored data points per site.

Fig. 1: Random sample for 10 sites (ordered by storage capacities) with jobs (left) and data stored (right).

$resourcetypes = \{A, B, C, D\}$: $\forall t \in resourcetypes : Pr(t) \in [0, 1]$, normalize such that $\sum_{i \in resourcetypes} Pr(i) = 1$ (PMF)

Jobs reads a data point of the same type:
$type \in \{A, B, C, D\}$ The location of a running job depends on its type and the provided capacities of the same type on sites based on probability mass functions (PMF) looking like the one in Fig. 2 (left). A job prefers local data since the workload balancer would rather choose a site with input data. This rate is built into the model with a 50 percent suppression of the branching factor of N (N sites). In other words, jobs aim at internal data with double chances.



(a) Job probabilities of one site. (b) Probabilities of global data types.

Fig. 2: A random sample of independent probabilities of local job types (left) versus global data types (right).

Data points are collections of files called datasets in the ATLAS use case. Such a data set comprises 1 to 100 files in the model to reflect dynamic dataset

sizes. Some popularity is assigned to datasets since there are differences in how likely a data set will be read. Data sets are from 100 users. Those users are in classes of job types and of activeness.
$type \in \{A, B, C, D\}$
$size \in [1, 100]$ files such that the median is 10 files.
user_id, which indicates the users submitting jobs.

5 Results and Conclusions

For replication factor 1, the method places the single copies of data to sites. Each data point is placed once. From this situation, for each number of sites in the network, replication runs are performed.

In the **first demonstration**, replicated data is added on all sites. Fig. 3 shows replication of different shares of the global data and for a different number of sites or nodes in the network. A replication factor of, e.g., 1.4 means replication of 2 on a 40 percent data share. The **second demonstration** includes the most extensive five sites providing free space for replicating data. The **third demonstration** covers feeding only the variable of *user_id* as data into the replication method. As in the section 4 on data described, there are 100 users, of which about ten are very active. These are like power users submitting lots of jobs. Users further submit jobs onto one or two data types. The results of the second and third cases are shown in Fig. 3.

Each run in the evaluation is repeated several times. System parameters for each run differ significantly, so the fluctuations from run to run are large. Many more simulation runs (per replication, per number of sites) would be necessary to obtain smooth curves.

6 Conclusion and Future Work

The method looks for the best cluster parameters on a 10 % sub batch and then uses the near-optimal solution on the full data. We observe only several percentage points decrease from a 100k batch to the entire data population of 1M. On a high-performance computer, applying the optimization would be feasible on a data batch of 1 million data points from 10 million data points in total. Complexity also depends on the dimensionality of data and the number of clustering types besides general model parameters and population size.

It is further shown that even on a weak predictor such as users, the method finds proper subsets to get closer to a near-optimum data replication and placement.

References

1. Boru, D., Kliazovich, D., Granelli, F., Bouvry, P., Zomaya, A.Y.: Energy-efficient data replication in cloud computing datacenters. Cluster computing **18**, 385–402 (2015)

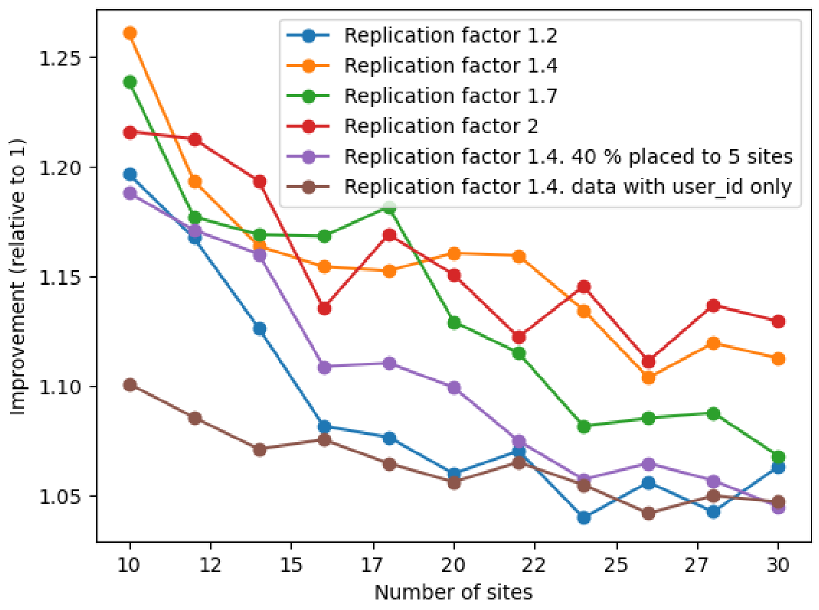


Fig. 3: Relative improvements on network metric depending on replication factors and the number of sites in the network.

2. Brezany, P., Mueck, T.A., Schikuta, E.: A software architecture for massively parallel input-output. In: Waśniewski, J., Dongarra, J., Madsen, K., Olesen, D. (eds.) Applied Parallel Computing Industrial Computation and Optimization. pp. 85–96. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
3. Casey, R.G.: Allocation of copies of a file in an information network. In: Proceedings of the May 16-18, 1972, spring joint computer conference. pp. 617–625 (1971)
4. Lamehamedi, H., Szymanski, B., Shentu, Z., Deelman, E.: Data replication strategies in grid environments. In: Fifth International Conference on Algorithms and Architectures for Parallel Processing, 2002. Proceedings. pp. 378–383. IEEE (2002)
5. Loukopoulos, T., Ahmad, I.: Static and adaptive distributed data replication using genetic algorithms. Journal of Parallel and Distributed Computing **64**(11), 1270–1285 (2004)
6. Vamosi, R., Lassnig, M., Schikuta, E.: Data allocation based on evolutionary data popularity clustering. In: Computational Science–ICCS 2018: 18th International Conference, Wuxi, China, June 11–13, 2018, Proceedings, Part I 18. pp. 153–166. Springer (2018)
7. Wolfson, O., Jajodia, S., Huang, Y.: An adaptive data replication algorithm. ACM Transactions on Database Systems (TODS) **22**(2), 255–314 (1997)