

# DynDTA: Proposal for a Dynamic Data Transfer Agent for the CMS Experiment

Björn Barrefors  
University of Nebraska-Lincoln  
bbarrefo@cse.unl.edu

June 9, 2014

## Abstract

The CMS experiment is a world-wide physics collaboration storing around  $O(50\text{PB})$  of data on more than 50 sites across 4 continents. Managing all this data is a manual job taking time from both physicist and data managers at each site. New data is expected to be generated by the end of 2014 further increasing the task. This proposal suggests and investigates an automated and intelligent data agent for data replication and deletion. Two equations are adopted to measure system-wide balance to respectively measure the effectiveness of storage and CPU utilization. We develop a daily system-wide storage and load balancing algorithm and provide measurements to prove its effectiveness.

## 1 INTRODUCTION

An increase in the amount of data generated by scientific experiments [cite] combined with an increased awareness of energy consumption [cite] the last years have increased the demand for more efficient data placement in large computing grids. By the end of 2014 a new set of experiments will start up at the LHC [cite]. All data is not equally popular [cite], there are datasets analyzed by several physics groups while other datasets are only of interest to a few scientists. Currently when a dataset is needed the group or scientist demanding it will request a replica to his or her site [cite]. But a new computing model is emerging where instead of transferring data to the jobs, jobs are transferred to the data to decrease bandwidth usage and data utilization [cite]. In this model a job can be executed almost instantly as there is no need to transfer up to almost 100TB of data. The model is not perfect, in case of a popular dataset the site(s) hosting replicas can become overflowed with jobs causing a CPU usage delay instead of transfer delay. A new data management policy is needed for popularity based dataset replication for optimal data and CPU availability. In this paper we investigate possible measurements of popularity and suggest a popularity based system-wide storage and load balancing algorithm. A series of metrics are suggested for measuring success in terms of storage and load balancing combined with costs of keeping a balanced system.

## 2 RELATED WORK

## 3 DATA MANAGEMENT ALGORITHM

### 3.1 Dataset Popularity

Creating a balanced system is of importance as it will both decrease the amount of data which just takes up space without being accessed and decrease the number of accesses on particular data which could cause overload on either network or CPU's. We have come up with a balance equation which can be continuously optimized to keep the system balanced. Using the standard deviation of all datasets ratio between number of accesses and number of replicas we can get a good measurement of the system balance. Combining equation 1 and 2 achieves this. Notice that the size of each dataset is added in to balance the equation based on dataset size. Here  $n\_access$  is the number of accesses during a time period  $t$ ,  $size\_GB$  is the size of the dataset, and  $n\_replicas$  is the number of replicas for the dataset.  $\mathcal{N}$  is the set of all datasets.

$$\delta = \int_0^t \frac{n\_accesses}{size\_GB * n\_replicas} dt \quad (1)$$

$$\sigma = \sqrt{\frac{\sum_{d \in \mathcal{D}} (\delta_d - \bar{\delta})^2}{|\mathcal{D}| - 1}} \quad (2)$$

The goal is to minimize equation 2, using a variation of frequency vectors we propose to use algorithm 1. Because the algorithm need to differentiate between increasingly popular and decreasingly popular datasets we introduce  $\delta_f$  as the change in frequency in equation 3,  $\delta_f < 0$  means the dataset have more then average replicas per GB and  $\delta_f > 0$  means the dataset have less then average replicas per GB. The closer to 0 the better. The algorithm uses exponential weight to avoid temporary spikes or drops in usage to trigger a subscription or deletion. We use a time window of  $w$  days, each day with a  $\delta_f$ , where days further away are given an inverse exponential weight. We also define  $\delta_{f+}$ ,  $\delta_{f-}$ , and  $available\_space$  in equations 4, 5, and 6.

$$\delta_f = \delta - \bar{\delta} \quad (3)$$

$$\delta_{f+} = \int_0^t \frac{n\_accesses}{size\_GB * (n\_replicas + 1)} dt \quad (4)$$

$$\delta_{f-} = \int_0^t \frac{n\_accesses}{size\_GB * (n\_replicas - 1)} dt \quad (5)$$

$$available\_space = total\_space * 0.95 - used\_space \quad (6)$$

### 3.2 Site Selection

Site selection is decided based on average available CPU. Since the dataset replicas deleted should have very low current usage we can use the state before current changes to make decisions. Available

---

**Algorithm 1** Dataset selection

---

**Require:**  $\mathcal{D}$ , the set of all datasets in AnalysisOps. Assume function `size()` exists for both dataset and set of datasets and returns total size in GB of dataset(s). Algorithm returns  $\mathcal{S}$ , the set of datasets to subscribe, and  $\mathcal{R}$ , the datasets to remove.

```
1:  $\mathcal{F} = \emptyset$ 
2: for all  $d \in \mathcal{D}$  do
3:    $\delta_f = 0$ 
4:   for 1 to  $w$  do
5:     Get  $\delta_f$  for the day, multiply by exponential weight and add to total  $\delta_f$ 
6:   end for
7:   Add tuple  $(d, \delta_f)$  to  $\mathcal{F}$ 
8: end for
9:  $\mathcal{S} = \emptyset$ 
10:  $\mathcal{R} = \emptyset$ 
11: loop
12:   Pop dataset  $s$  with highest  $\delta_f$  such that  $\delta_f > 0$  and  $\delta_{f+} > \overline{\delta_f}$ .
13:   if not  $s$  then
14:     Break loop
15:   end if
16:   if  $\mathcal{S}.\text{size}() + s.\text{size}() > \text{budget}$  then
17:     Break loop
18:   end if
19:    $\mathcal{S} = \mathcal{S} \cup \{s\}$ 
20:   while  $(\mathcal{R}.\text{size}() + \text{available\_size}) < \mathcal{S}.\text{size}()$  do
21:     Pop dataset  $r$  with lowest  $\delta_f$  such that  $\delta_f < 0$  and  $\delta_{f-} < \overline{\delta_f}$ .
22:     if not  $r$  then
23:       Break loop
24:     end if
25:      $\mathcal{R} = \mathcal{R} \cup \{r\}$ 
26:   end while
27:   if  $(\mathcal{R}.\text{size}() + \text{available\_space}) < \mathcal{S}.\text{size}()$  then
28:      $\mathcal{S} = \mathcal{S} \setminus \{s\}$ ; Break loop
29:   end if
30: end loop
31: return  $\mathcal{S}, \mathcal{R}$ 
```

---

CPU is calculated using equation 7 where  $\alpha$  is the sum of all CPU hours from the top three days during the last three weeks and  $\beta$  is the number of CPU hours used yesterday.

$$available\_cpu = \frac{\alpha}{3} - \beta \quad (7)$$

Sites with the most available CPU hours will be selected for new replicas. Of datasets selected for deletion the replica with the least number of accesses will be chosen for deletion. More restrictions is needed to avoid oversubscription to one site, using simulation to better understand effects of the currently proposed algorithm will be of great importance and is described in the following section.

## 4 SIMULATION AND TESTING

### 4.1 Simulation

It is important to understand the effects of these algorithms and find thresholds for when to subscribe and delete datasets before deploying them in a real system. A simulation testbed needs to be deployed to observe behaviours such as disk churn, network traffic, and performance. Evaluating results from a testbed will show flaws and strengths in the different parts of the dynamic data manager.

### 4.2 Testing

This section describes measurements used to evaluate the dynamic data manager.

- Number of unused datasets. A balanced system should decrease the number of unused replicas, if a replica is not used it should be deleted. If this number doesn't go down then there is an issue with the dynamic data manager.
- Minimize the median waiting time if there are sufficient available CPU's anywhere in the system. We want to place data in such a manner that the system is utilized as much as possible at all times. We are also interested in the median and not average as users will always be able to do weird things which affect the system but cannot be accounted for, such as submitting 10000 jobs to an unpopular dataset which only exists at a site which is currently fully utilized. A case which would be almost impossible to catch before it happens. The measurement of this test is expressed in equation 8.

$$Q_t = \frac{\sum_{d \in \mathcal{D}} \frac{\sum_{j_d \in \mathcal{J}_d} qt}{|\mathcal{J}_d|}}{|\mathcal{D}|} \quad (8)$$

Where  $\mathcal{D}$  is the set of all datasets,  $\mathcal{J}_d$  is the set of all jobs for dataset  $d$ , and  $qt$  is the queue time of job  $j_d$  while there was sufficient amount of CPU's available in the system. Where an optimal value for  $Q_t$  would be 0. Such a value would be virtually impossible to achieve, an acceptable value will have to be defined.

## 5 SCHEDULER INTEGRATION

To fully utilize a dynamic data management system it is important to have a tight coupling with the job schedulers. This will both provide instant popularity information about datasets for the agent and make it possible for the scheduler to quickly react to changes in data allocation. This functionality will become available with CRAB3.

## References