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

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

May 29, 2014

Abstract

The CMS experiment is a world-wide physics collaboration storing around O(50PB) of data on more than 50 sites across 4 continents. Manage all this data manually to make sure the correct data is available at the right time and freeing up space as datasets are no longer needed can be a challenge. We therefore suggest an automated and intelligent data transfer agent to keep track of dataset activities, making decisions on when and where to replicate datasets and when to remove them when no longer needed. Our suggested solution is to calculate a frequency vector based on number of accesses per replica and even out the differences by adding and removing replicas.

1 Introduction

The CMS experiment is a world-wide physics collaboration storing around O(50PB) of data on more than 100 sites across 4 continents. Data is organized into datasets where each dataset have some common properties making it likely to be completely used in analysis jobs. Available storage at participating sites is currently managed by data managers for different physics groups, if a dataset is needed for analysis a replica is subscribed to that site. Such a solution would perform very well on a system with infinite amount of bandwidth, storage, and CPU power. Since such a system doesn't exist we suggest a change in how the CMS experiment manages jobs and data. Instead of moving large amounts of data to where the job is submitted jobs can be pushed to sites where data is already available. If there are CPU's available at any site a submitted job should be able to run, an automatic dynamic data manager would be responsible for replicating popular datasets and removing unpopular datasets to fully utilize both storage and CPU's across the system. Such a manager consists of three major tasks, finding popular datasets early in the popularity period, selecting an optimal site for load balancing, and finding datasets which are no longer popular and can be removed. In this paper we will talk about possible algorithms for solving each of these tasks and what is needed in order to develop such algorithms. We will also propose metrics to measure the successfulness of the agent.

2 Related Work

A similar project was done for the ATLAS experiment [PanDA] which is one of the other major experiments at CERN. Similarly to our dynamic data placement their goal was to improve the efficiency of storage and reduce queue time. PanDA was developed as a dynamic data manager to integrate their data distribution with the global job scheduler. PanDA uses a fairly aggressive algorithm to quickly react on jobs in the global queue and was developed based on ATLAS's requirements and data popularity distribution. Several lessons can be learned from the development of PanDA, one of the things they did well was to analyze data popularity for use in the dynamic data manager. Another was to closely integrate the data manger with their job scheduler for fast reaction both in terms of acknowledging increase in popularity of datasets and to quickly adjust the scheduler to system changes. Even though ATLAS have noticed improvements since the deployment of ATLAS there have also been significant increase in network traffic. Furthermore their algorithm is still relatively novice and greedy as it focus a lot on the current state of the queue and less on creating a balanced system.

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 continiously 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.

$$\delta = \int_0^t \frac{n\_accesses}{size\_GB * n\_replicas} \, \mathrm{d}t \tag{1}$$

$$\sigma = \sqrt{\frac{\sum\limits_{d \in \mathcal{D}} (\delta_d - \overline{\delta})^2}{|\mathcal{D}| - 1}} \tag{2}$$

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 δ_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 δ_f, where days further away are given an inverse exponential weight. We also define δ_{f+}, δ_{f-}, and $available\_space$ in equations 4, 5, and 6.

$$\delta_f = \delta - \overline{\delta} \tag{3}$$

$$\delta_{f+} = \int_0^t \frac{n\_accesses}{size\_GB * (n\_replicas + 1)} \, \mathrm{d}t \tag{4}$$

$$\delta_{f-} = \int_0^t \frac{n\_accesses}{size\_GB * (n\_replicas - 1)} \, \mathrm{d}t \tag{5}$$

$$available\_space = total\_space * 0.95 - used\_space \tag{6}$$

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.

$\mathcal{F} = \varnothing$
for all $d \in \mathcal{D}$ **do**
  $\delta_f = 0$
  for 1 **to** w **do**
    Get δ_f for the day, multiply by exponential weight and add to total δ_f
  end for
  Add touple (d, δ_f) to F
end for
$\mathcal{S} = \varnothing$
$\mathcal{R} = \varnothing$
loop
  Pop dataset s with highest δ_f such that $\delta_f > 0$ **and** $\delta_{f+} > \overline{\delta_f}$.
  if not s **then**
    Break loop
  end if
  if $\mathcal{S}$.size() + s.size() > budget **then**
    Break loop
  end if
  $\mathcal{S} = \mathcal{S} \cup \{s\}$
  while ($\mathcal{R}$.size() + available_size) < $\mathcal{S}$.size() **do**
    Pop dataset r with lowest δ_f such that $\delta_f < 0$ **and** $\delta_{f-} < \overline{\delta_f}$.
    if not r **then**
      Break loop
    end if
    $\mathcal{R} = \mathcal{R} \cup \{r\}$
  end while
  if ($\mathcal{R}$.size() + $available_space$) < $\mathcal{S}$.size() **then**
    $\mathcal{S} = \mathcal{S} \setminus \{s\}$; Break loop
  end if
end loop
return $\mathcal{S}, \mathcal{R}$

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

$$available\_cpu = \frac{\alpha}{3} - \beta \tag{7}$$

Sites with the most avaialable 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.

$$Q_t = \frac{\sum\limits_{d \in \mathcal{D}} \frac{\sum\limits_{j_d \in \mathcal{J}_d} qt}{|\mathcal{J}_d|}}{|\mathcal{D}|} \tag{8}$$

  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