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

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

April 28, 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 treat this as a "catching heavy-hitters on a sliding window" problem [Braverman et. al].

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. Furthermore even though ATLAS have noticed improvements since the deployment of ATLAS there have also been significant increase in network traffic. In our project we want to improve on PanDA's idea with even better data popularity projection and decrease in popularity without putting too much pressure on the network infrastructure.

3 Goals

The ultimate goals for a dynamic data transfer agent is to decrease queuing times of CMS jobs while improving storage and CPU utilization, these are hard to measure goals so we have come up with other more specific and measurable goals to better test successfulness.

- Minimize the fluctuation of users or accesses per replicas for each dataset. We need to define a popularity model for datasets which would decide whether to use number of users or accesses. We express this as minimizing the following equation:

$$\Delta = \int_0^t \left| \frac{n\_users}{n\_replicas} \right| \mathrm{d}t$$

  The optimal value for Δ will have to be found using heuristics.

- 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. We express this as minimizing the following equation:

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

  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.

4 Methods

4.1 Popularity Model

Making predictions about popularity demands a good understanding of what popularity is and what components describes it. We currently have three years worth of data about dataset usage available in the Popularity Database [Reference]. We will use this data to find a popularity model and decide which parameters are of interest for selecting both datasets and sites. The result is to be used for a more accurate data selection algorithm for both replication and deletion.

4.2 Catching Heavy Hitters

To achieve the above goals we want to recognize as datasets are becoming popular at an early stage. Such a problem can be expressed as a "catching heavy-hitters" problem. Example of such a problem is to find the most popular queries on Google during a day/week/month/year etc [reference]. Companies like Google often deal with massive amounts of data making streaming algorithms popular to analyze data, we will not be dealing with data on that scale and does therefore not need to use an approximation algorithm but the general process is the same. Finding heavy hitters in a data stream is based on the ability to find the Euclidean distance between two frequency vectors [reference]. Using two same sized sliding windows positioned at different times we can build a frequency vector for each where each element represent the popularity of a dataset, this could be number of accesses, number of users, CPU hours, etc during the time window. Heavy hitters can be found by calculating the Euclidean distance between the vectors and pinpoint the element(s) which is causing the distance. Depending on the magnitude of change a dataset can then be flagged as having a sudden increase in popularity or decrease in popularity. Using real world heuristics is suggested to determine the optimal time window and minimum change to be considered a heavy-hitter for both increase and decrease in popularity.

4.3 Site Selection

Once a heavy-hitter has been found the agent will need to find either a target site or site to remove from depending on increase or decrease in popularity. Such an algorithm will need to have knowledge about the current state of the system, including site utilization, calculated replica popularity of datasets at site, network infrastructure, and available storage. I suggest that each site keeps a database with this information which is updated daily. When a site is to be selected the central agent will query all sites about their current state getting back a value which represents how fit the site is to receive a dataset of popularity magnitude calculated at the previous stage, or in the case of removal the need to free up resources, a selection of site is then based on the values returned from each site.

4.4 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 [reference].

References