Journal of Physics: Conference Series **513** (2014) 042042 doi:10.1088/1742-6596/513/4/042042

Analysis and improvement of data-set level file distribution in Disk Pool Manager

Samuel Cadellin Skipsey[1], Stuart Purdie[2], David Britton[1], Mark Mitchell[1], Wahid Bhimji[3], David Smith[4]

[1]Department of Physics and Astronomy, University of Glasgow, G12 8QQ, UK
[2]University of St Andrews, School of Computer Science, KY16 9SX, UK
[3]University of Edinburgh, School of Physics & Astronomy, James Clerk Maxwell Building, The Kings Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK
[4]European Organization for Nuclear Research (CERN), CH-1211 Genève, Switzerland

E-mail: samuel.skipsey@glasgow.ac.uk

Abstract. Of the three most widely used implementations of the WLCG Storage Element specification, Disk Pool Manager[1, 2] (DPM) has the simplest implementation of file placement balancing (StoRM doesn't attempt this, leaving it up to the underlying filesystem, which can be very sophisticated in itself). DPM uses a round-robin algorithm (with optional filesystem weighting), for placing files across filesystems and servers. This does a reasonable job of evenly distributing files across the storage array provided to it. However, it does not offer any guarantees of the evenness of distribution of that subset of files associated with a given "dataset" (which often maps onto a "directory" in the DPM namespace (DPNS)). It is useful to consider a concept of "balance", where an optimally balanced set of files indicates that the files are distributed evenly across all of the pool nodes. The best case performance of the round robin algorithm is to maintain balance, it has no mechanism to improve balance.

In the past year or more, larger DPM sites have noticed load spikes on individual disk servers, and suspected that these were exacerbated by excesses of files from popular datasets on those servers. We present here a software tool which analyses file distribution for all datasets in a DPM SE, providing a measure of the poorness of file location in this context. Further, the tool provides a list of file movement actions which will improve dataset-level file distribution, and can action those file movements itself. We present results of such an analysis on the UKI-SCOTGRID-GLASGOW Production DPM.

1. Introduction

By definition, a distributed filesystem must implement a mechanism for allocating individual files to one or more of the underlying storage volumes abstracted by it. Design of the file distribution mechanisms may be constrained by one or more requirements; maintaining even distribution of files across all volumes, maintaining proportional use of the space in each volume, managing IO and network load on individual servers, and so on. The Disk Pool Manager[1, 2] (DPM) Storage Element implements a simple file distribution algorithm, in which a static list is used to sequence the allocation of incoming files to disk server filesystems. The static list defaults to an unweighted round-robin list of all filesystems, but administrators can provide weightings (which result in duplicate entries for those filesystems in the list) in order to (for example) distribute more files to larger storage volumes. While this mechanism does evenly distribute files across


Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd

the available storage, it does not guarantee good properties for any higher level grouping on those files. For example, many VOs group their files into "datasets", representing a complete set of files associated with a particular analysis or production run on the grid. Generally, future work will be performed on an entire dataset, rather than one or two files considered in isolation. Storage Elements cannot be aware of the existence of datasets, nor the particular memberships of such higher-order groupings that any incoming files have. It is therefore not clear *a priori* that any given dataset will be evenly distributed across filesystems in a DPM (or any other storage element). (This problem is much more significant for file-oriented distributed filesystems: block-distributed systems, such as HDFS[3], or striped systems, such as Lustre[4, 5], spread per-file load over multiple servers, effectively smearing out hotspots from higher-order inhomogeneities.)

Contributory evidence for the case that not all datasets are well distributed on DPM disk servers is represented by the load "brownout" phenomenon that large DPM installations appear to suffer from to various extents. Sites in the UK and elsewhere with large fractions of their storage resource dedicated to the ATLAS VO experience occasional load "brownouts", in which a small fraction of their disk servers experience instantaneous request loads high enough to effectively overload their network connection (figure 1 shows an historical example of this as captured by monitoring at UKI-SCOTGRID-GLASGOW). The fact that this load only affects a minority of disks at any time suggests that the affected disk servers have more interesting files (and thus a larger share of the files in the currently hot datasets at the site) than the others.

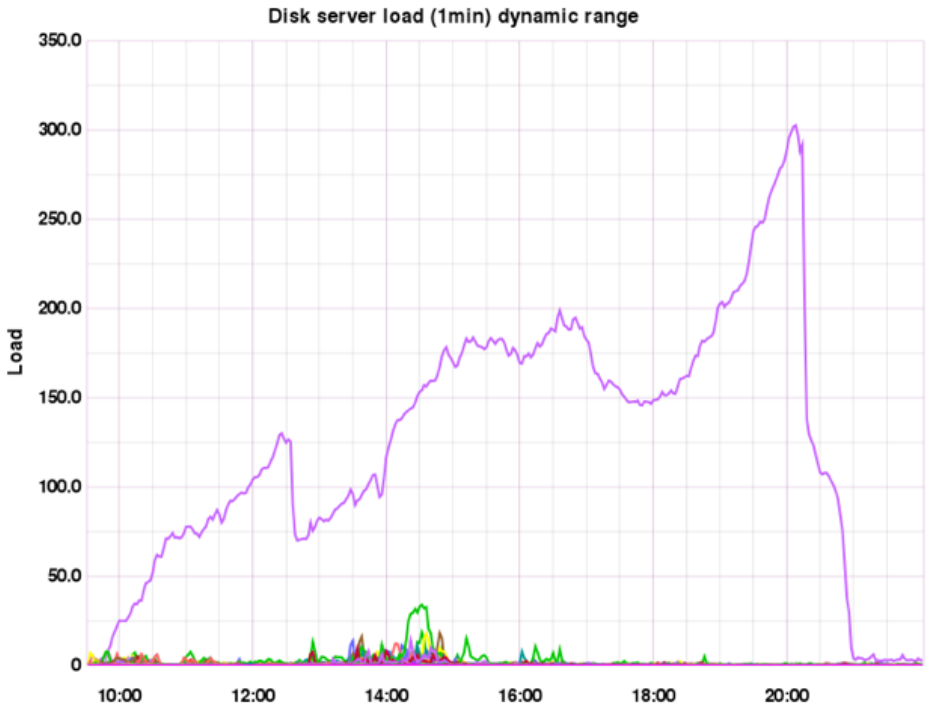

Figure 1. Example of single-disk anomalous catastrophic load due to unbalanced dataset distribution (taken from monitoring[6] at UKI-SCOTGRID-GLASGOW, August 2012). Pink line is a single disk server's reported load (all other disk servers are plotted, but are barely visible on this scale).

In the absence, at present, of effective load management tools for any of the popularly used data transfer protocols in DPM, a reasonable approach is to attempt to analyse the distribution of datasets at a site in order to "smooth" the load out over the entire ensemble.

1.1. Dataset Identification

In order to rebalance at the dataset level, we must identify the files in each dataset. For many experiments, this is easy, as the dataset is identified with the logical directory containing a set of files. This approximation is used for the analysis presented in this paper, and we use "directory" interchangeably with "dataset" for the remainder of the text.

For ATLAS' Rucio[7] system, this identification is broken (Rucio abstracts its internal logical file system at the storage element layer, storing files in a hierarchical hashed directory structure instead). In this case, we must query Rucio itself to determine the datasets and their contents. The Rucio DMLite plugin[8] could be used for this purpose, and is being investigated by the authors for future extension of the tool.

2. Defining 'unbalanced'

Ideally, we want a measure of balance that gives a progressive indicator, and thus allows us to pick out the 'least balanced' directories. This has the advantage of enabling work to be focused on the areas where it will have largest effect. The measure used is to determine the number of files in the most subscribed filesystem for any given DPNS directory, and divide that by the mean; where the mean is obtained by taking the total number of files in the directory and dividing by the number of filesystems containing files that are members of it. This measure ignores cases where there are empty filesystems - for our purposes, we are concerned with the peak load, so empty filesystems are only a problem as a side effect - it's the over loaded ones that matter. (Other metrics, of course, can replace the described measure, allowing the rest of the engine to balance according to different requirements, such as redistributing files to even out disk usage across disk servers of varying age.)

This figure of merit is a dimensionless number, and is mostly independent of the number of files in the directory. In practice, it is poorly conditioned (varies widely for a small change) for directories with a small number of files; but they are much less of a problem, so it's not unreasonable to ignore the balance of directories with fewer files than some threshold (where 20 was used in the implemented code). Figure 3 is a representation of such an analysis on the UKI-SCOTGRID-DPM production DPM.

Unfortunately, when applied to the task of perfectly balancing the file distribution, this measure produces results which are hard to tune, especially as the number of moves increases. Instead of pursuing this approach, we wrote code to find the unbalance factor for the Nth worst directory. This gives a way to gradually improve the balance on the storage, starting from the worst case, and do it in a reasonably smooth way. The idea is to launch the code with a target of 2 (i.e. considering issues up to the second worst case); let it move files to make the worst as balanced as the second worst, and then iterate with increasing thresholds until file distributions are improved to our satisfaction (the stopping criterion being the number of iterations we wish to select - there was no set cutoff limit applied). This iterative algorithm has the additional advantage that single iterations can be run periodically to maintain the balance of a DPM filesystem, with reduced overhead compared to periodically scheduling a complete rebalancing.

2.1. Building work lists

Now that we can identify directories with balance problems, we can pick out the worst, and rebalance those. This is done by building a list of all files to be moved, and then passing that off to a rate-balancing engine. The first algorithm tried here was to do random assignment of moves between least and most subscribed filesystems.

Simulation (running the balancing algorithm against a database copied from the production DPM, but not performing the actual file moves themselves, then post-hoc generating a graph of the new file counts) showed that this appeared to produce reasonable results. In rare cases where the random assignment process left datasets in less than perfect balance (due to statistical

variation), the next run of the balancer resolves the issue. Such events are expected to be very rare, although they may crop up more with frequent execution in the maintenance case.

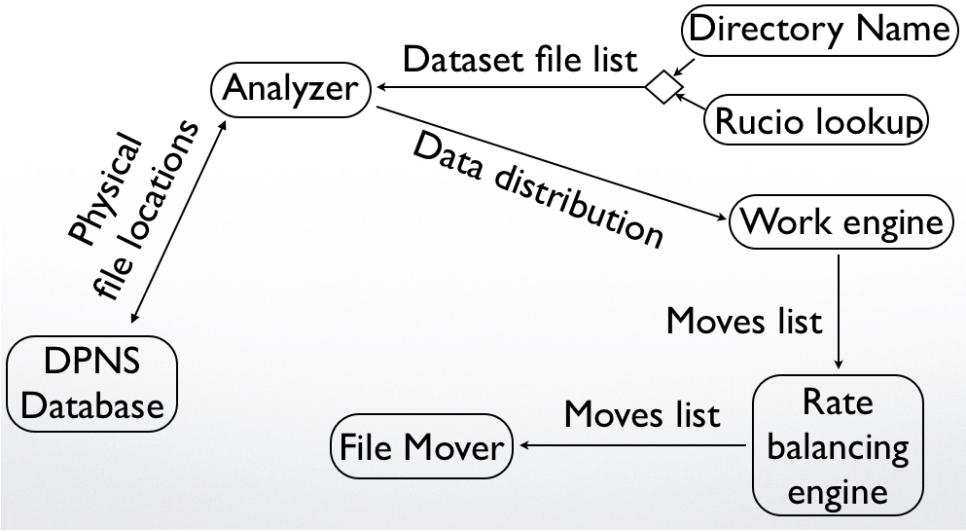

Figure 2. Indicative flowchart of implemented algorithm.

2.2. Rate-balancing engine
This ensures that no more than some specified limit of transfers is executed simultaneously. As part of this process, we also shuffle the order of moves slightly so that the same source and destination are not adjacent in the move list repeatedly, reducing load spikes from the move process. This part is written to be modular, but algorithms other than the initial design have not been developed.

3. File Placement in Reality
Application of the work lists to a real world DPM instance was hindered by the fact neither DPM nor DMLite expose "raw" file placement in their APIs, especially the Python and Perl versions. While you can trigger creation of a file replica at any time, there is no way to specify the destination (instead, destination selection passes through the same round-robin selection that all file creation operations use).

At the time of submission, this functionality was an active feature request against DPM [9] . By the time of publication, the first iteration of the functionality had been provided (via the davix interface), but there was no time to produce useful results.

4. Conclusions
The problem of efficient file distribution in distributed filesystems has dimensions beyond the simple problem of equalising capacity use across member servers. In the case of DPM, we have identified correlations between asymmetric filesystem load and poor distribution of files within a given dataset. We have presented a candidate algorithm for improving this distribution, almost orthogonal to the actual distribution of numbers of files on each disk. While this algorithm could not be tested on real systems, due to pending functionality in DPM, the initial simulated results do produce improved evenness of file distribution at the dataset order. In addition, the provided algorithm is sufficiently modular that it could be applied to balancing of files on other metrics

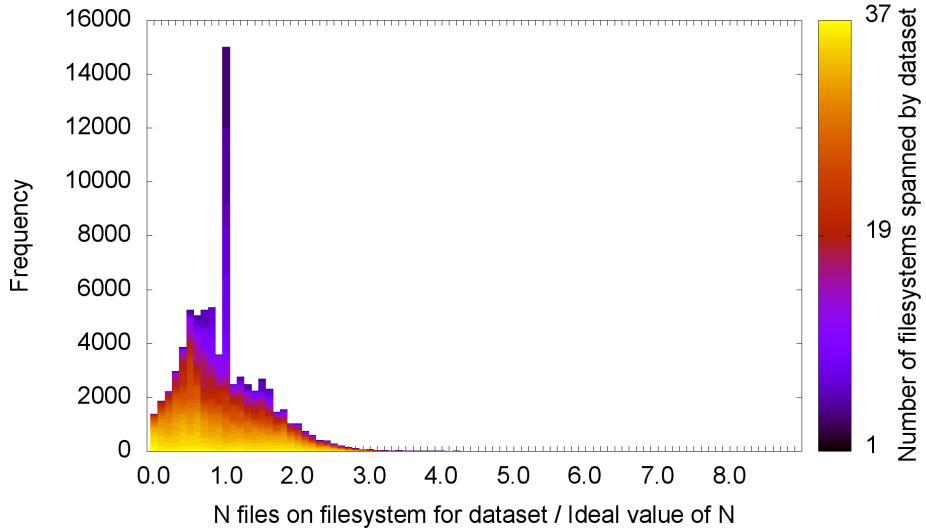

Figure 3. Stacked histogram of normalised dataset distribution (per filesystem), coloured by total number of filesystems in a given dataset's span. Value 1.0 corresponds to a filesystem with the precisely 1/n of the files in the dataset, where the span is n. It is clear that for small spans (purple), the distribution is good, while as the span increases, the distribution quickly becomes wide and unbalanced. For some datasets, a small number of filesystems exist with up to 7.4 times the average number of files, although their number is small enough that they are not visible on this graph directly. The chosen range of the x axis indicates the actual length of the long tail.

(from the lowest order, "number of files per server", to potentially more esoteric measures, such as disk server bandwidth, or topological concerns). We expect to be able to produce real life tests in the coming months as targeted replication becomes available in the davix interface.

References

[1] Frohner Á *et al* 2010 *J. Phys.: Conf. Ser.* **219** 062012 doi:10.1088/1742-6596/219/6/062012
[2] *The DPM Project site.* https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm
[3] *HDFS Design Documentation.* http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Large_Data_Sets
[4] *Lustre OpenSFS Community* http://lustre.opensfs.org/
[5] Wang F *et al* 2009 "Understanding Lustre Filesystem Internals" http://users.nccs.gov/~fwang2/papers/lustre_report.pdf
[6] Crooks D *et al* 2014 "Monitoring in a grid cluster" This proceedings.
[7] *The Rucio project* http://rucio.cern.ch
[8] Van Dongen D 2013 http://cds.cern.ch/record/1602854/
[9] *Tracking ID for DPM Targeted Replication API* https://its.cern.ch/jira/browse/LCGDM-1007