Abstract. Recent developments, including low power devices, cluster file systems and cloud storage, represent an explosion in the possibilities for deploying and managing grid storage. In this paper we present how different technologies can be leveraged to build a storage service with differing cost, power, performance, scalability and reliability profiles, using the popular storage solution Disk Pool Manager (DPM/dmlite) as the enabling technology.

The storage manager DPM is designed for these new environments, allowing users to scale up and down as they need it, and optimizing their computing centers energy efficiency and costs. DPM runs on high-performance machines, profiting from multi-core and multi-CPU setups. It supports separating the database from the metadata server, the head node, largely reducing its hard disk requirements. Since version 1.8.6, DPM is released in EPEL and Fedora, simplifying distribution and maintenance, but also supporting the ARM architecture beside i386 and x86_64, allowing it to run the smallest low-power machines such as the Raspberry Pi or the CuBox. This usage is facilitated by the possibility to scale horizontally using a main database and a distributed memcached-powered namespace cache. Additionally, DPM supports a variety of storage pools in the backend, most importantly HDFS, S3-enabled storage, and cluster file systems, allowing users to fit their DPM installation exactly to their needs.

In this paper, we investigate the power-efficiency and total cost of ownership of various DPM configurations. We develop metrics to evaluate the expected performance of a setup both in terms of namespace and disk access considering the overall cost including equipment, power consumptions, or data/storage fees. The setups tested range from the lowest scale using Raspberry Pis with only 700MHz single cores and a 100Mbps network connections, over conventional multi-core servers to typical virtual machine instances in cloud settings. We evaluate the combinations of different name server setups, for example load-balanced clusters, with different storage setups, from using a classic local configuration to private and public clouds.

1. Introduction

In the age of big data and petascale computing, energy efficiency is becoming more and more important. The density of computing power has increased enough for computing centers to be limited not by space, but by the available power. Locations for large computing infrastructure projects are chosen on the basis of the surrounding climate to minimise effort and cost for cooling. So while the computing power, the available memory, and storage density are continuously increasing, the energy consumed turns out to be a major constraint for higher-performance computing or clusters, making it important to investigate raising energy efficiency.

One way to increase the energy efficiency of a computing center is by consolidation of services on a virtualised infrastructure. Another is to invest in specialised hardware to increase the
efficiency of single applications. For storage clusters virtualisation is not appropriate, as the I/O devices, the most precious resource, should not be shared with other applications. On the other side, memory and computing power are often overdimensioned on these systems and are idle a significant part of the time. However, most storage systems can be split into one part responsible for the metadata handling and another one for the data storage, the former being a good target for virtualisation.

Another way to increase efficiency for computing is through better utilisation of the available processing hardware with vectorisation, optimisation for cache efficiency or redesigning code for multicore architectures. Further along this line is the use of different architectures like the Xeon Phi and GPUs, which provide more performance for the same power consumption, or ARM-based devices, which provide slightly less performance for much less power consumption. Higher performance is achieved by scaling out.

For storage, the ARM-based approach is very attractive for the aforementioned reason that computing on the storage devices is rather over- than underdimensioned. While the ARM server market is still very immature, it makes sense to research in the area how storage clusters would perform on such an architecture.

In this work we present research on the implications of virtualisation and the utilisation of ARM-based devices on the performance of the grid storage manager DPM (Disk Pool Manager). A large refactoring of DPM [2] now enables it to make use of multiple storage backends, such as HDFS, S3-based storage or Ceph and simply provides the grid compatibility layer of top of them. This allows us to focus on the metadata part of the storage, which we can virtualise efficiently. The move in the build and release system to EPEL and Fedora also provides us with automatic builds for the ARM architecture, allowing us a relatively simple way to evaluate our system on ARM. As a challenge, we chose to use the Raspberry Pi as a minimal platform for a very low-power and low-cost DPM metadata server. The results should serve as a baseline of what we can expect on such ARM devices.

In the following, we show an introduction to the DPM architecture after the refactoring and possible configuration scenarios. We then explain the test setups used for the performance measurements along with the results. In the 4th section we attempt an evaluation of the setups in form of performance per Watt and especially how the more exotic configurations fare with respect to traditional systems. In the last section, we conclude this paper with a summary of our findings and an outlook for the future.

2. An Overview of DPM and Deployment Scenarios

The DPM is a grid storage manager widely used in the WLCG. As opposed to dCache [11] and Castor [12], it does not manage tape storage, which makes it used mostly at Tier 2 sites. The largest installation today is 2 Petabytes. DPM exposes multiple protocols, including HTTP, Xrootd, GridFTP, and support the Grid Security Infrastructure with authentication through X509 client certificates and VOMS extensions.

Traditionally, a DPM system consists of a metadata server, the head node, which is attached to a MySql or Oracle database, and a number of disk pools to provide storage. The task of the metadata server is to accept client requests, perform the authentication/authorisation procedure and then to redirect to one of the disk nodes.

In the last two years DPM has undergone a major refactoring effort which, beside giving a clean, modern C++ codebase to work on, provides much more flexibility to extend the capabilities. The now underlying plugin-based library dmlite [3] provides an abstraction to the database, pool management and I/O layers, so that plugins for S3 protocol based backends, hdfs [?] backends, and a caching layer in front of the database could be implemented. The S3 backend plugin [4] enables the metadata server to redirect clients directly into the Amazon cloud or to a local storage cloud running OpenStack or the Ceph Rados Gateway. In these cases, the
client is then limited to access over http. In the case of the hdfs plugin [5], all DPM-supported protocols are available, but a gateway module has to be installed on the hdfs nodes. Similarly one can integrate Ceph through the Rados Block Device.

The integration of the hdfs namespace shows the power of the database abstraction. Instead of using a dedicated database, DPM can link into the hdfs namespace, thus seamlessly integrating the hdfs system and avoiding consistency issue that other approaches have. The issue when using a separated database for an additional frontend is that files ingested through the native backend interface are not directly visible in the new frontend’s namespace. More importantly, the same goes for files being deleted through the native interface.

The dmlite memcached plugin [6] is another example of the database abstraction and the capability of the dmlite library to stack multiple interface implementations on top of each other. It creates an in-memory cache of the namespace and of the locations of the file replicas (spread over the disk nodes) to decrease the latency for a result from the metadata server. As a side-effect, the network traffic to the database, if it is remote to the metadata server(s), is largely reduced, facilitating horizontal scaling of the metadata server while keeping the database server the same. The dmlite library also enables us to use off-the-shelf software for frontends, as has been shown by the http frontend based on apache2 and mod_dav. This is explained in a bit more detail in the paper Towards an HTTP Ecosystem for HEP Data Access [1]. For more detailed information about the dmlite plugins and frontends, please refer to our paper at CHEP 2012 [2].

This architecture makes DPM very useful as a thin layer between already existing services and the grid functionality provided to the clients. In the following we will describe the configuration vectors that will be analysed in the next section.

2.1. Deployment Scenarios

The configurations used in the analysis show the two major dimensions of scaling a computing system: scaling up and scaling out. For scaling up, we provide three different platforms: the Raspberry Pi on the low end, virtual machines running on OpenStack with a single core, and 4-core Intel Xeon server on the high end. As comparison we also measured the performance on a medium instance on Amazon EC2. The ability to scale out will be analysed by replicating the number of Raspberry Pi- and virtual machine-based metadata servers, which connect to a single database, and connect to a DNS load balancer. For the physical machine, storage is local and remote, the Raspberry Pis and the VMs connect to separate storage VMs. More information about the characteristics of the machines used can be found in table 1.

These setups are much simplified by the facilities given in the CERN computing center: the OpenStack installation gives us much flexibility to start new instances, either for additional metadata servers, storage, or a metadata database. Additionally, CERN provides a Ceph storage cluster for testing, which we have already successfully used in combination with DPM.

In the test configurations, we made use of the DNS load balancer and deployed the databases on OpenStack. Since we only tested the metadata performance, we did not make use of the Ceph storage cluster. Nevertheless, this environment is what we expect to find in computer centers in the near future, and definitely when using a commercial cloud provider.

3. Measuring the Performance of Different Configurations

In the following we describe the test setup and the performance metric used to analyse the performance of DPM in the different setups described above. In order to limit ourselves, we decided to start the evaluation with a single protocol, HTTP, and a single metric that describes most of the user interactions with a metadata server, the stat request.

For the tests, the performance testing utility Perfsuite [7] has been used, which also provided the metadata test programs. The tests were run from a physical machine with a 1 Gbps network
connection. In the cases where load balanced metadata servers were tested, multiple machines of this type were used.

We focus on HTTP, because we see the growing importance of HTTP/WebDAV in the High Energy Physics community and because it is the only available protocol that unifies file access, transfer, list and data management in one protocol. This also facilitates the testing, since many of these interactions translate into the same commands. File listing and metadata access, the main task of the metadata server, translates into the WebDAV PROPFIND command. For this request, it is extremely important to handle a high number of requests for a large number of user simultaneously. The tests measure the processing time of a single requests as well as a rate of requests per second for multiple users. This describes the latency a single user experiences and the throughput that the platform can provide.

As a reference we also provide latency numbers for the two other important requests, GET, and HEAD. These requests provide the redirection to one of the disk nodes for actual file access. The HEAD request is an alternative to PROPFIND reading the file metadata, such as size and creation data, from the disk node instead of from the metadata server. Since we are only interested in the performance of the metadata server, our client does not follow the redirection to the disk node for GET or HEAD. Another aspect is the PUT of files, which we will not consider in the experiments: one issue is that the implementations can vary widely for different backends, another is that the volume of read accesses is much larger than write accesses.

The results can be seen in table 2. We have compared the durations of the requests between a system serving only one client and another one also serving 100 clients in the background at maximum capacity. The measurements are taken on the client side, so they include low level system latencies and network latencies. The results show that in the former case the request durations between the low end and high end machines are small, but that we can see significant differences if the systems are under high load.

| Type             | 1 client | 100 + 1 client |
|------------------|----------|----------------|
|                  | PROPFIND | HEAD | GET | PROPFIND | HEAD | GET |
| Raspberry Pi     | 0.06     | 0.27 | 0.41 | 15.89     |       |     |
| OpenStack VM     | 0.02     | 0.13 | 0.19 | 0.12      | 0.28  | 0.53|
| Phys. Machine    | 0.004    | 0.05 | 0.05 | 0.01      | 0.06  | 0.06|

Table 2. The mean durations of a PROPFIND, HEAD, and GET requests as measured on the client. A - indicates that the value was not measured.

This is consistent with the findings of the scalability test. Smaller systems reach the peak number of users earlier, and when they do, clients experience an increase in latency for every request. For the results of table 3 we chose an “optimal” value of users, after which the
throughput did no longer increase. It has to be noted that although the systems operate at
their maximum capacity, the throughput is not degrading (much) with more concurrent users.
For the deployment of small system it is still important to manage the number of users accessing
a single machine by load balancing.

| Requests/sec | Raspberry Pi | OpenStack VM | EC2 medium VM | Phys. Machine |
|--------------|--------------|--------------|---------------|---------------|
| 1 instance   | 25.8         | 394.18       | 162.27        | 1701.23       |
| 3 instances  | 71.3         | 1087.0       | -             | -             |
| scaling      | 2.76         | 2.75         | -             | -             |

Table 3. Throughput of PROPFIND request on the optimal number of users. For the Raspberry
Pi we used 30 clients and for the rest 100 clients. Performance does not fall with higher numbers,
but stays steady. A - indicates that the value was not measured.

The load balancing scenario is also shown in table 3. It can be seen that the systems scale
very well, although not perfectly. This can be accounted at least partly to the database load
and the network traffic to the database. As our experiments run in a virtualised environment,
it is difficult to estimate the actual bandwidth available. During the tests we found out that the
load balancer provided by CERN is not matching the test scenario very well. Since our clients
are virtual and only distributed along three physical machines, DNS caching would prevent the
load balancing. Also the balancing did not happen fast enough for our test scenarios with tests
usually taking less than a minute. The results here thus represent a “perfect” load balancer
which we emulated by targeting each client machine against a different server.

In order to see how the influence of the database can be limited, we performed another test run
with the memcache plugin enabled shown in table 4. The results show that the throughput can be
increased by almost the factor two and that the bandwidth needed between the metadata servers
and the shared database is reduced. This affects the VM setup much more than the Raspberry
Pi, as the lower performance of the Raspberry Pis never creates much network traffic.

| Requests/sec | Raspberry Pi | DB network throughput | OpenStack VM | DB network throughput |
|--------------|--------------|------------------------|--------------|-----------------------|
| direct mysql | 25.8         | 1.3 Mbps               | 394.18       | 108 Mbps              |
| memcached    | 58.47        | 0.6 Mbps               | 764.02       | 32 Mbps               |

Table 4. The influence of the memcached plugin on network traffic.

In the next section we will discuss how these performance results relate to the costs of the
different systems, but first we will look a bit more closely at the reasons for these results, especially
the ones for the Raspberry Pi.

The Raspberry Pi has three potential bottlenecks: the network I/O, the CPU, and memory.
Network performance can be limited either by the throughput of only 100 Mbps and by additional
latency coming from the connection of the network hardware to the CPU via USB. In our case,
none of these seem to apply: the maximum throughput is with 4.8 Mbps well under the practical
throughput limit of the Raspberry Pi of 24 Mbps. The limitations of the Raspberry Pi are thus
in the CPU, memory, or a combination of both. An analysis of the CPU load over time suggests
that it limits the system, although load levels on the VMs also rise very high to more than
100. A throughput test of memcached suggests that it can handle far more transactions than
performed, but this is only a part of the CPU-memory interaction.
Even without further investigation it is safe to say that we are looking forward to devices with a similar power profile as the Raspberry Pi, but a higher core frequency and/or multicore processors. Such devices are on the one hand consumer devices like the CuBox [8] and the O-DROID [9], on the other hand first ARM-based servers as the Boston Viridis [10].

3.1. Developing for the Raspberry Pi
As already mentioned, the development process was relatively straight-forward, since the Fedora project, through which we release our software, already provides us with ARM builds for ARMv5 and ARMv7. The Raspberry Pi CPU implements ARMv6 and can run binaries built for ARMv5. In an effort to make use of the hardware floating points units and other optimisations we did not go to the official repositories, but to a fork, which made it a bit more complicated, since we had to compile some packages ourselves. Except for the fact that builds usually take several hours, there were no issues with the process to build and create RPMs for ourselves. A related experiment also showed that Raspberry Pis can be integrated in a Java-based build environment (Atlassian Bamboo) if the timeouts for builds are increased.

The fact that rpms are available of course does not mean that the software will work flawlessly. Here the ARM architecture provides a challenge and opportunity to reveal bugs in the software, which could not have been detected on the x86 architecture. In our case it was the use of a dangling pointer: on x86, the memory was still valid, in the case of ARM, it was not. Thanks to the findings, we could create a patch from which future DPM releases will profit.

4. Evaluating the Efficiency and TCO
To bring the performance results into perspective, we will evaluate two aspects of the different machines tested: cost and power consumption. Considering a storage provider that has the requirement of delivering a steady rate of 500 requests per second over the next 3 years, we can attempt to suggest a system configuration that is energy- and cost-efficient. We take the ARM-based approach, and the deployment in a private or public cloud into account, to focus on the “non-traditional” systems.

The goal is to minimise the total cost of ownership (TCO) for the provider, which consists of many aspects. We concentrate on the buying price of the hardware and the power consumption over time, since all other factors are heavily dependent on the surroundings in a computing center, such as the cost to add a device to the network or the cost of maintenance personnel. Another aspect to consider is the probability of failure and the resulting cost of replacing a device. This factor is very likely to affect the Raspberry Pi more than the other configurations as it is a consumer device. The Raspberry Pi results should thus be seen as a proxy for energy-efficient ARM servers, which might have similar performance characteristics.

First we will compare the energy-efficiency between virtual machines and the Raspberry Pi, representing future deployment scenarios. Table 5 shows the performance per Watt. It is a very promising result that a minimal consumer device is already in the same order of magnitude as a virtualised server environment. Minor changes in the ARM setup, e.g. a dual core processor, might get the performance per Watt on the same level with the VMs. Also systems like the Boston Viridis already demonstrate how to encapsulate such a system into a rack-friendly enclosure.

To get a more complete picture of the TCO for the use case mentioned above, we will calculate the power consumption over three years and the number of devices needed to achieve 500 requests per second. The results are shown in table 6. With our data, the Raspberry Pi fares a bit better than the VMs in overall cost, but 25 Raspberry Pis are needed instead of 2 VMs. Estimating that the cost of attaching one Raspberry Pi to the computer center infrastructure is more costly than the device itself, this scenario becomes much less attractive but, again, can be used as a proxy for ARM-based server systems.
Table 5. Performance per Watt The power consumption of the Raspberry Pi is its max. consumption. The power consumption for the VMs is, as in 1, estimated from the rack consumption and split by core.

|                                      | Raspberry Pi | OpenStack VM | EC2 Medium VM |
|--------------------------------------|--------------|--------------|---------------|
| Requests/sec                         | 25.8         | 394.18       |               |
| Consumption                          | 3.5 W        | 30 W         |               |
| Throughput/Watt                      | 7.37         | 13.1         |               |

Table 6. Costs for sustaining 500 requests per second over three years. All prices are in Euro, estimated additional cost for infrastructure are 100 Euro for local machines.  

1 The calculation assumes that the server is 70% under load and consumes only 20% of the nominal power during the rest of the time. So the machine consumes $c = Consumption \times 24 \times 365 \times 3Wh$; we calculate with $c \times 0.7 + c \times 0.3 + 0.2$.  
2 The electricity price is assumed at 7 Cents/kWh, which is a reasonable price in France.  
3 The last row multiplies the estimated cost over 3 years by the number of machines needed for 500 requests/sec, assuming good scaling. Both setups over provision a little. A - indicates that this field does not apply, e.g. you do not pay the power consumption of an EC2 machine explicitly.

The costs of 3 Amazon instances to achieve the same results are far off from the others, but one has to consider that they include the costs of surrounding infrastructure as well as the cost for replacement and hardware maintenance. If the difference matches the cost in a computing center can hardly be generalised.

5. Conclusion
In this paper we have tested the grid storage manager DPM on different platforms in order to demonstrate that it is flexible enough to be used in on small as well as large deployments and to assess low-power ARM devices for their suitability for storage. For this we conducted a number of performance tests to calculate the performance per Watt for each configuration.

The results show that even systems as small as the Raspberry Pi can be used to manage the metadata server of a grid-scale storage system, of course with some trade-offs in performance. The more limited number of concurrent users per device can be countered by scaling out, simply depending on a high-performance database in the background. To facilitate this and reduce the database load, local caching on the metadata servers can be used.

Our experiences with the Raspberry Pi have been very positive and suggest a potential for upcoming ARM-based servers for storage systems. They are primarily suited for metadata activities, but could, if it is possible to attach mass storage, also be useful for the storage devices themselves. In alignment with other activities researching ARM-based servers for computation, we await the release and mass-manufacturing of the next generation 64 bit ARM processors and will be continuing our feasibility studies.
References
[1] Towards an HTTP Ecosystem for HEP Data Access Fabrizio Furano, Adrien Devresse, Oliver Keeble, Martin Hellmich, Alejandro Álvarez Ayllón CHEP2013
[2] DPM: Future Proof Storage Alejandro Álvarez Ayllón, Alexandre Beche, Fabrizio Furano, Martin Hellmich, Oliver Keeble, Ricardo Rocha CHEP2012
[3] Dmlite https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm/Dev/Dmlite
[4] Dmlite S3 https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm/Dev/Dmlite/Plugins/S3
[5] Dmlite HDFS https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm/Dev/Dmlite/Plugins/HDFS
[6] Dmlite Memcache https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm/Dev/Dmlite/Plugins/Memcache
[7] Perfsuite https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm/Admin/Performance
[8] CuBox http://www.solid-run.com/products/cubox
[9] O-Droid http://www.hardkernel.com/renewal_2011/main.php
[10] Boston Viridis http://www.boston.co.uk/solutions/viridis/viridis-2u.aspx
[11] dCache http://www.dcache.org
[12] Castor http://castor.web.cern.ch