A STRATEGY FOR COST EFFICIENT DISTRIBUTED DATA STORAGE FOR IN-MEMORY OLAP

Olga Mordvinova[12], Oleksandr Shepil[1], Thomas Ludwig[3], Andrew Ross[1]
[1]SAP AG, [2]Heidelberg University, [3]Hamburg University/DKRZ
{olga.mordvinova,oleksandr.shepil,a.ross}@sap.com; ludwig@dkrz.de

ABSTRACT

With the availability of inexpensive blade servers featuring 32 GB or more of main memory, memory-based engines such as the SAP NetWeaver Business Warehouse Accelerator are coming into widespread use for online analytic processing (OLAP) of terabyte data volumes. Data storage for such engines is often implemented in standard storage technologies like storage area network (SAN) or network attached storage (NAS) with high hardware costs. Given the access pattern, storage costs can be reduced by using a distributed persistence layer based on commodity architecture. We discuss an example of an in-memory OLAP engine with a focus on storage architecture. We then present an implementation of a distributed persistence layer that is optimized for the access pattern of such engines. Finally, we show the cost-saving potential and discuss the performance impact compared to SAN systems.

KEYWORDS

Storage issues, Distributed storage, Cost efficient storage, OLAP

1. INTRODUCTION

Systems for online analytic processing (OLAP) need to handle growing volumes of time-critical data and also meet challenging user requirements for fast response times and flexible support for complex or ad hoc queries. Relational databases are still the best-known and most widely implemented back-end architecture for OLAP systems. Given the growing demand for processing large data volumes, such legacy systems often give rise to maintenance and optimization issues arising from their use of materialized views as well as long access times to the external storage devices.

Modern memory-based OLAP engines are designed to meet these challenges. By holding and processing data in main memory, such engines can execute complex or ad hoc queries over large volumes of structured data and reliably deliver subsecond response times. Such systems are usually installed on high-end servers attached to reliable high-performance storage technologies like network attached storage (NAS) or storage area network (SAN), as in Greenplum or Kognitio systems, connected using high-performance computing (HPC) network technology. The storage and its network are major cost drivers for an OLAP appliance. For this reason, commercial solutions are moving toward usage of commodity architecture. The Netezza data warehouse appliance offers a solution based on Fibre Channel (FC) and low-cost server blades, and the Vertica column-based data store can run on either SAN or commodity storage infrastructure.

The state-of-the-art approach embodied in the SAP NetWeaver Business Warehouse Accelerator (BWA) offers one or more orders of magnitude improvement in speed and flexibility of access to the data. However, implemented on SAN infrastructure with HPC network technology, it faces the cost challenge described above. To meet this challenge, we have developed a concept for reliable distributed storage that reduces the up-front cost of a BWA solution by running it on commodity hardware.

This paper is organized as follows. In Section 2, we introduce the BWA OLAP engine and its storage architecture and outline the challenge. In Section 3, we review the latest concepts in the field of distributed storage. In Section 4, we introduce our proposed concept for distributed storage and explain how to adjust it to the specific architecture of the BWA engine. In Section 5, we show that our approach does indeed offer a cost-efficient realization of time-critical OLAP without loss of performance. In Section 6, we summarize our results and conclude with an outlook on future development.

2. A COLUMN-ORIENTED OLAP ENGINE

The SAP NetWeaver BW Accelerator (BWA) is a leading example of an in-memory column-oriented online analytic processing engine for use in business warehouse and business intelligence applications (Fig. 1). It aggregates large volumes of structured data held in main memory and executes queries over the data fast enough for real-time scenarios. It is designed for deployment in the IT cluster of any company that stores large and growing volumes of business data in a standard relational database.

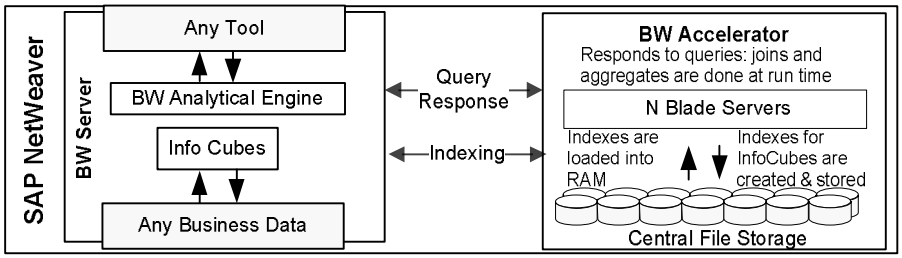

Figure 1. OLAP engine architecture

The following main architectural features contribute to the high performance of the BWA. Data is highly compressed in main memory to eliminate the runtime cost of disk access and its structures are optimized for sequential read. Tables of data are split vertically into separate columns, each listing values for one attribute. This is similar to the techniques used by C-Store [Stonebraker 2005], MonetDB [Boncz 2008], or Vertica [Stonebraker 2007], and contrasts with the traditional approach of splitting tables into rows and keeping all the attribute values of a row together. Tables with many rows are split horizontally into partitions, each with some of the rows, and the partitions are processed in parallel. The shared-nothing parallelism approach reduces a query to a graph of atomic operations, many of which are performed in parallel without overhead for synchronization. The multi-server architecture allows scalability, reliability, and load balancing.

To ensure predictability, the OLAP solution is released as preconfigured software on certified hardware. Such an appliance can be upscaled with additional blade servers or upgraded to next-generation servers. The storage volume required scales almost linearly [Legler 2006] with main memory capacity and number of cores. Currently, BWA blades are equipped with 64-bit Intel Xeon processors, up to 32 GB of RAM per blade, and an external SAN subsystem attached via FC and running a cluster file system such as Global Parallel File System (GPFS) [Schmuck 2002]. However, BWA hardware costs are at present too high for many mid-market customers. Much of the cost arises from the central storage. The current BWA architecture does not use local storage for data. Every blade in an appliance includes 70 GB to 150 GB of local storage, which currently stores only the operating system, cluster administration software, configuration data, and swap partitions, so only a fraction of the available space is utilized.

In summary, the current BWA uses high-cost external storage without exploiting the inexpensive local storage in the blades. We have developed a new persistence layer (PL) approach in which the application data is stored on the local hard disks and the appliance itself runs as a Beowulf cluster.

3. RELATED WORK

The idea of inexpensive commodity storage has been explored in the industrial (Google FS, Hadoop FS, FARSITE, Vertica [Stonebraker 2007]), scientific (Ceph, XtreemFS [Hupfeld 2007], OceanStore [Rhea 2003]), and high-performance (Lustre) communities. Even if motivated by different requirements or functionality aims its distributed persistence on commodity hardware have some aspects in common.

(1) *Software based fault tolerance*. As commodity hardware often fails, the software must implement service redundancy, e.g., by duplicating services for data or metadata management. Lustre pairs cluster nodes as active and passive, which creates no overhead [Braam 2002] since a passive server for one pair can be active for another. In the Google File System (GFS) [Ghemawat 2003] and its open source implementation Hadoop Distributed File System (HDFS) [Venner 2009], the metadata server is mirrored by a shadow

metadata server, which is activated only in case of failure. Multiple metadata servers ensure performance and reliability.

(2) *Metadata is separated from data* and handled in separate processes. Lustre [Braam 2002] provides an asymmetrical system architecture with one metadata server and multiple object storage servers. There is an exclusive cluster node for the metadata server process in GFS/HDFS. As a rule, the metadata server also handles, e.g., time stamps, locking, load balancing, and data distribution in the cluster. The GFS master contains the mapping table for all chunks with information about their size, location, and mapping to the appropriate file in the landscape, and initiates chunk rebalancing if necessary. In Ceph, the metadata server adapts its behavior to the current workload dynamically by metadata replication across multiple servers [Weil 2006a]. Namespaces in Farsite are stored separately in a hierarchical tree structure that allows different namespace roots. Several servers controlling namespace roots are summarized in a directory group. As Farsite is a decentral network file system for a huge number of heterogeneous hosts [Adya 2002], it has to handle an insecure infrastructure, i.e., hosts that may crash without warning or leak information to third parties. To ensure metadata integrity in a directory group, it makes use of the Byzantine Agreement protocol. The metadata of a directory group with R_D group members is considered as valid if not more than $(R_D - 1)/3$ members are erroneous. The extremely wide-area storage system OceanStore [Rhea 2003] uses cryptographic approaches to ensure data integrity in an insecure environment.

(3) *Data is chunked* into predefined minimum data units. Clients do not deal with the block layer and leave the mapping to the separate processes or data server nodes. The chunk size reflects the application. Since GFS/HDFS needs to handle only a few access patterns, the chunk size is 64 MB. Such big chunks need less metadata in main memory and reduce network traffic [Ghemawat 2003]. OceanStore [Rhea 2003] handles data objects that contain the object version and a 160 bit globally unique identifier (GUID), which is hashed (via SHA-1) from the user's public key and the file name and content. Such data objects can be fragmented variably and distributed over the cluster nodes. During each data request, the reference on the root GUID is returned, which contains references on GUIDs of the next level and so on, an approach similar to the inode concept used in UNIX file systems.

(4) *Data is held redundantly*. This offers fault tolerance and overcomes the performance bottleneck of central storage, since overall bandwidth is limited by the network and not by the central storage controller. The challenge here is data placement to achieve reliability and optimize performance. GFS places replicas on different racks and uses MapReduce [Dean 2004] and BigTable [Chang 2006]. In Ceph, the data arrangement function CRUSH ensures reliability by splitting the cluster into failure domains [Weil 2006b]. In OceanStore, data is moved constantly (is "nomadic") to ensure that it is stored on the hosts where it is needed. This approach is challenged by limited network throughput for frequent data movement: [Bindel 2000] proposes "promiscuous" caching and "introspective" monitoring to meet the challenge. In Vertica, data is replicated in the cluster in projections [Stonebraker 2007], which are in effect materialized views. Each projection contains a subset of the columns of a table in a particular sort order, so that a group of columns can be sorted in one order on one node and in another order on another node, which can improve query performance.

We studied all these design trends for commodity storage architecture when developing our PL appliance. We chose a new approach because the other approaches are either proprietary (GFS, Lustre), do not provide predictable performance (XtreemFS, Ceph, Hadoop), or do not suit our needs (OceanStore, FARSITE).

4. DESIGN OVERVIEW

We aimed to develop off-the-shelf distributed storage with good failure handling. Where possible, we reused the existing application infrastructure and worked with the existing data structures and access patterns. To optimize performance, we integrated the distributed persistence into the application rather than building a POSIX-compliant file system.

4.1 Application Infrastructure

The BWA distributed server architecture separates data and metadata processing [Legler 2006, Ross 2009]. A separate index server process handles all operations on indexes. As an index server acts as a client for the persistence layer, the persistence layer client can be implemented in the index server. The BWA name server

process supervises the whole cluster environment and manages availability. A name server process runs on each cluster node and provides read access to metadata stored in central shared memory. For reliability, there are three write-enabled name servers. A primary name server writes and the other two are shadows, ready to take over in case of failure. The metadata manager for the persistence layer benefits from this mechanism if it is in the name server.

In the BWA, the minimum logical units of operation are indexes [Legler 2006]. These are organized hierarchically in namespaces and contain content files (transactional data), temporary index files (uncompressed indexes), attribute files (index columns), and configuration files.

Typically, many midsize files (30–50 MB) containing millions of documents are created and written at once. The only small files (2–5 KB) are configuration files. Once written, files are not modified frequently. There are two types of modification: indexing (data is loaded into an index) and optimization (the index is compacted). During indexing, the data is stored in temporary files. During optimization, the data from the temporary files is read in parallel and written in the attribute files. Each partition of the attribute data is handled by a different server, but with evenly distributed data all the servers perform I/O at the same time. This peak load can approach the limits of advanced storage solutions. The workloads primarily consist of large (≥1 MB) sequential writes (appends) and reads; seeks within a file are rare.

We optimized the distributed PL for large sequential writes and reads. Since all operations are performed at the index level, files are kept together below a defined level in the hierarchy. We ensured consistency for concurrent reads of a file by using multiple clients, but with only one writer per index we did not have to deal with concurrent appends to the same file.

4.2 The Distributed Persistence Layer

The PL architecture includes a separate data server process for data operations, metadata management integrated in the name server process, and a client integrated in the index server process (Fig. 2).

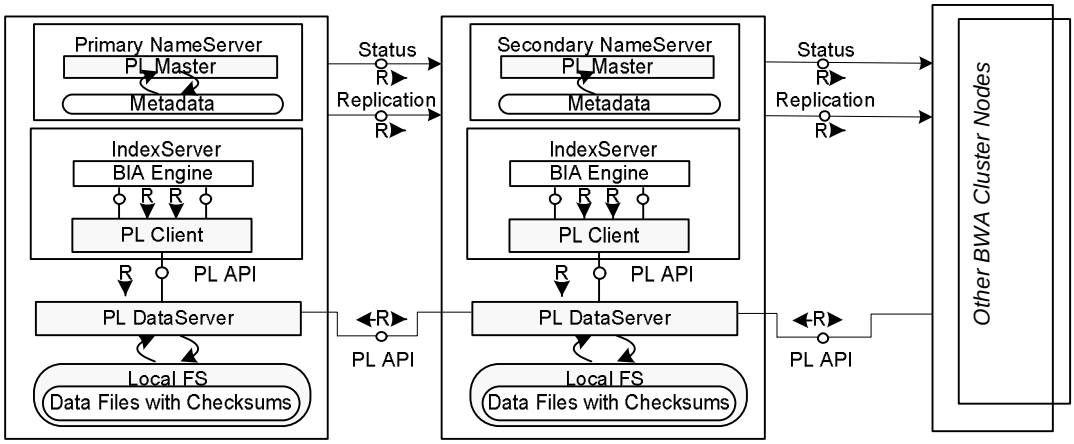

Figure 2. Distributed persistence layer architecture

The PL client is implemented as an index server library, which communicates with the local or remote PL master for metadata operations and local or remote data servers for data operations (Fig. 3). If an index replica is stored locally on the host, the client uses the local file system API for reads.

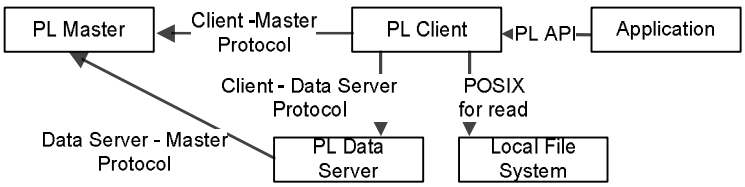

Figure 3. Outline of interactions in persistence layer

The PL master handles index metadata and operations on it, i.e., index creation, deletion, renaming, copy, and index replication. There are three types of metadata: index metadata, index structure, and runtime information. Index metadata includes index name, index state, and index-to-host mapping. Runtime information includes, e.g., the status of data servers and data load distribution. The PL master stores index metadata and runtime information in a tree structure in local shared memory. This "topology" tree is stored to disk and periodically replicated as a snapshot to all servers in the cluster. The shared memory on cluster nodes reduces the network traffic for metadata read operations. To ensure consistent metadata, only the primary PL master can modify the topology tree. Index structure data (index directories, directory hierarchy, and file metadata) is stored and handled by the data server.

During index creation or recovery, the PL master identifies the most suitable hosts for replicas. Replicas are placed on hosts with the lowest disk space utilization and with the lowest (estimated) workload. This estimation is based on the total number of indexes on the host, the number of recently created replicas, and the role of the index server (master or backup). We assume that the master index server has a higher work load than the backup index server. To ensure fast access to index data and minimize network traffic, the first replica is always stored locally on the host of the indexing client. Three replicas normally provide sufficient redundancy, but this number can be customized. At I/O failures or during periodic health checks, the master deletes or creates index replicas and directs the data server.

The PL data server executes the data operations requested by clients or by other data servers. We use pipelining as in [Ghemawat 2003] for efficient data distribution. The PL data server also performs background activity such as replication and garbage collection. All the data for a given index are placed on the same host and replicated at this level. In case of failure, one index replica allows continued work. Other replicas can be recovered asynchronously later.

Consistency and Data Integrity: An index is considered as consistent if the client always sees the same data, regardless of which replicas it reads. Index data consistency is achieved by synchronous modification of all replicas. If an operation on a replica fails, the replica is marked as corrupted. Corrupted replicas are not operated on or replicated. Additionally, the data server validates data by creating checksums while writing (for each file block of 512 B) and verifying them during reads. If checksum verification fails, the data server returns an error to the client. The PL master marks the index replica as corrupted and the client reads another replica. For the checksums we use the CRC-32 algorithm, which offers both reliability and performance.

Replica Recovery, Replica Migration, and Garbage Collection: The PL master initiates index replica recovery as soon as the number of available replicas falls below a defined replica level. The master chooses a new replica target the using the strategy described above. Data recovery is then performed asynchronously, index by index. While an index is being replicated, all write requests to that index are blocked on the master until the replication is completed.

Replica migration is bounded with distribution of the active index servers that performs indexing and so needs its data. Since the role of the active index server changes during the application work, replica migration can be initiated asynchronously to ensure that data is stored on the hosts where it is needed. To ensure consistency, we first create an additional data copy on the target host. The master later deletes the surplus replica on the host from which the new replica was copied.

A garbage collection mechanism for recognizing and deleting obsolete replicas is implemented on the PL master and PL data server. Every data server sends a list of local indexes both at startup and regularly thereafter to the master. If the replica is not listed on the PL master, the data server deletes it. To identify obsolete index metadata, the master diagnostic thread compares the replicas listed in the main index leaf with replicas listed in the data server leaf. For any index metadata that lacks index data, the PL master deletes the topology reference.

Fault Tolerance and Reliability: As part of the name server infrastructure, the PL master has two roles: writer (or primary) and reader (secondary). If the writer becomes unavailable, a reader has access to the replicated metadata and takes over the primary's function. The engine continues to run as long as at least one master is available.

A PL data server considers an operation as successful if it succeeded on at least one data server, otherwise the client reports an error. We distinguish the following PL data server failure scenarios: (1) complete host is unavailable, (2) errors on execution of methods, and (3) server crash. In case (1), the PL master identifies the host during the regular availability checks and removes it from the topology. In case (2), the error is reported

to the client and the client reports the failed replica to the PL master. This replica is then considered as corrupt. If a read fails on a single data server, the client re-executes the method on the next known replica. If a write fails on a data server, it is not re-executed if a write on at least one data server was successful. In case (3), the client or a peer data server detects a broken network connection and the operation is repeated.

A PL client failure is either an index server crash or a crash of the whole host. The PL master detects a host crash during the availability checks and excludes it from any operation. When the host returns online, garbage collection is initiated.

5. EVALUATION

In this section, we compare the existing BWA central storage (CS**)** solution with a distributed PL solution. The main criteria for the comparison are total (capital and operating) cost and query processing performance (since query processing is the main function of the application). We also briefly consider the indexing performance, although this is usually not critical for an OLAP engine.

5.1 Total Cost of Ownership

The total cost of ownership (TCO) of a BWA solution comprises capital and operating costs [Ferrin 2002]. Capital costs include the initial acquisition cost plus any subsequent cost for extending the appliance to meet growing demand. The achievable savings per installation depend on the size of the cluster and how easily it can be upscaled or upgraded.

As an example to illustrate the possible reduction in capital cost using the new PL solution, we consider a small Hewlett-Packard cluster system for BWA. Table 1 shows its main components. This cluster provides 1164 GB of gross disk capacity (SAN), 64 GB of RAM, and 32 CPU cores. Each blade also has 364 GB internal disk capacity (2 hard disks each with 36 GB capacity and 2 with 146 GB) used for operating system and monitoring software. Implementing redundancy for the CS by using RAID with one-disk redundancy, we obtain an unformatted capacity of 1019 GB and with two-disk redundancy 874 GB. The total capital cost for this setup is €138 000, where €119 000 is for hardware and €19 000 for software, including FC and SAN monitors and licenses for Novell Suse Linux Enterprise Server (SLES). There are no further costs for the Oracle Cluster File System (OCFS), since it is included in SLES. For an IBM cluster we would need to add GPFS license costs.

Table 1. Cluster components of a small BWA setup

Number	Component
4	HP ProLiant BL460c G1 with 2 x quad-core Intel Xeon E5345 CPU at 2.33 GHz and 8 GB RAM
1	M5314C FC drive enclosure with 8 x 146 GB 15K dual-port disk drive
1	HP DL380 G5 base storage server
1	EVA4100 2C1D array with 8 x HP StorageWorks 146 GB 15K FC HDD
2	StorageWorks SAN switch 4/8

An appliance based on the PL has no costs for high-end technologies like SAN and FC. To achieve a storage capacity with internal disks comparable to the 874 GB above, we need to add a fifth blade. Here we assume that the five pairs of 36 GB disks are used for operating system and the five pairs of 146 GB disks for data to give 728 GB effective storage capacity with single redundancy (half the total data disk capacity). With this configuration, we can save €64 000 on the whole solution (a cost reduction of 53%), of which €57 000 is for hardware and €7 000 for software. The additional node (€5 000) adds 20% more CPU and RAM capacity to the BWA cluster.

To extend the storage capacity in a CS scenario, we need to install not only additional storage disks but also new server blades so as not to degrade BWA processing performance. If the requirements exceed the capacity of the storage server, we need to consider upgrading the SAN. Using the PL, the storage extension is achieved simply by adding server blades (€3 500). In the example, the server rack used has space for 16 half-height storage blades, so a further rack is needed for more than 16 blades. Figure 4 shows the cost increase for the two storage approaches for a storage enhancement from 500 GB to 3 TB. In both cases, the cost

increase is linear. There is no step discontinuity in the graph, since for 3 TB we need neither an SAN upgrade nor an additional server rack (at most 13 server blades are used). The absolute difference between the hardware costs for the PL and the CS solutions increases slightly with growing capacity. The greatest percent cost reduction (60%) using the PL is achieved with the smallest cluster (500 GB). The saving falls to 35% for a 3 TB cluster.

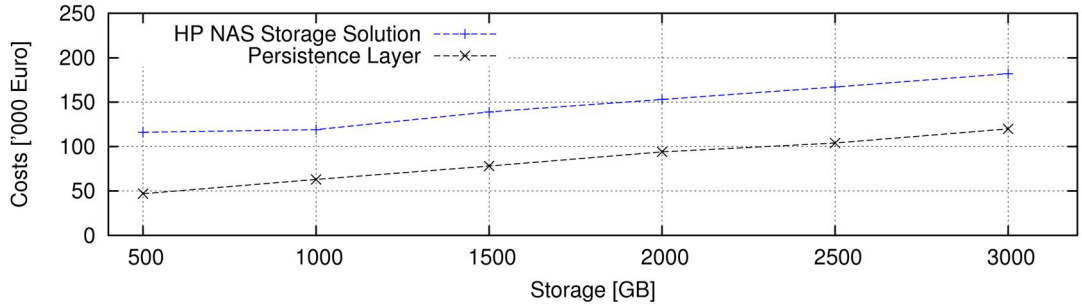

Figure 4. Hardware costs relative to storage scale up for the CS and the PL approach

Operating expenses include outlay for software and hardware maintenance as well as energy costs. In the evaluated package, the maintenance costs were €13 400 for 3 years. With PL we can save €9 500 (70% of total maintenance costs) support costs for FC switches, FC HDDs, and rackable storage and networking, even including maintenance expenses for an additional blade server (€400). This standard maintenance price per server will not be considered here more in detail. We also assume here this price remains the same, because, being integrated into existing central BWA maintenance tool cluster configuration, maintenance, and management remains transparent for the BWA system administrator and does not need extra effort.

The absence of storage infrastructure also affects the energy costs. The disk array in this configuration has an average power consumption of 640 W and the filer head about 410 W. Thus, the overall power savings are at least 1 kW. Adding four worker blades (at 500 W per blade) we reach about 3.1 kW for the whole HP cluster. The power consumption for the PL depends only on five worker blades and is on average 2.5 kW. So we reduce the power consumption by 20% compared to the CS solution. We do not take the FC switches into account because we could have used an Ethernet switch for high availability and network throughput.

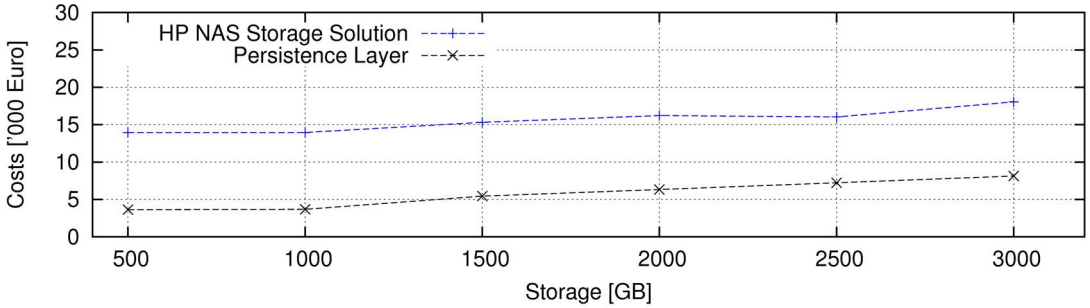

Figure 5. Operating costs relative to storage scale up for the central storage and the PL approach

We calculated the annual energy costs per year C_A using the formula $C_A = E \cdot 24 \cdot 365 \cdot P$, where E is the average power consumption in kW when the system is in use, 24 is the number of hours per day, 365 is the number of days per year, and P is the energy price per kWh (here taken as €0.18). Using the CS infrastructure, the calculated annual running cost was €4 900, whereas for the PL it was only €4 000.

Figure 5 shows the operating costs for both solutions when the storage is upgraded to 3 TB. Combining energy costs with maintenance costs, the PL solution remains more cost-efficient than the CS solution. The operating costs for the small system running on the PL (storage volume up to 1.5 TB) are over 70% lower. With growing storage volumes, the difference between the CS and PL installations decreases continuously, from 65% for 1.5 TB down to 55% for 3 TB.

Since a greater TCO reduction is achieved in smaller implementations, the PL approach is more suitable for small and midsize customers. For BWA installations with more than 13 blades, a CS solution may still be appropriate.

5.2 Performance

The aim of the first evaluation is to show the suitability of the PL approach for the existing BWA appliance. We evaluated the total performance impact by replacing the central storage with the PL infrastructure on a cluster of six nodes as described above. The performed application level tests are based on real data sets.

Since the BWA is designed for query processing, we focused on read performance. We imported 4 575 indexes with a total size of 105 GB, which used 315 GB disk space in the PL with three replicas. The data and the test queries were based on real customer data and varied in complexity. Total duration of all tests was about 3 hours. During the tests, the index data was loaded from the hard disk into the memory where the search was performed. Since in both cases the index search is done in RAM, we expected only a marginal performance difference. To check that the results were consistent, we repeated each test ten times. Figure 6 shows the results as the performance factor f between the PL and the CS, where $f = t_{PL}/t_{CS}$. The tests are numbered on the x-axis of the graph. Factor $f = 1$ indicates the same query performance, $f < 1$ better performance, and $f > 1$ worse performance on the PL.

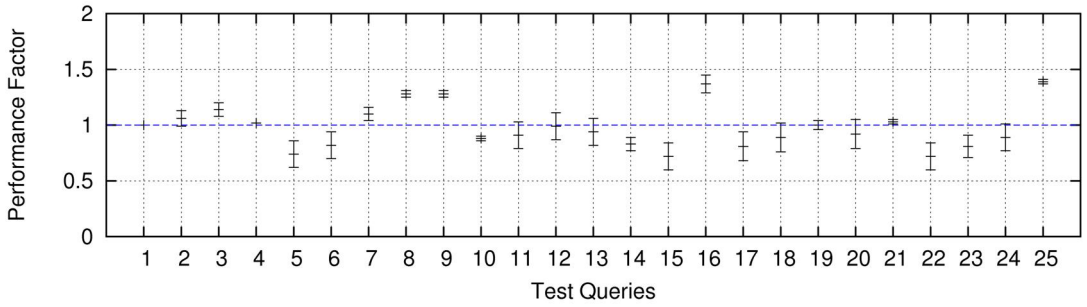

Figure 6. Query performance comparison between CS and PL with a test set of 25 queries. The error bars (95% confidence intervals) are small for measurements with low variance

The total performance factor for all 25 query tests was about 1, indicating that in most tests the search performance is unchanged. Variations of the performance factor are explained by data placement on the PL. The PL was faster when the index data was read from the local disk and slower when it was read over the network from another host. However, in the CS scenario, the data was always read from the SAN.

Indexing is the most challenging operation for the PL, so we evaluated it briefly. To reproduce a typical BWA scenario, we indexed a cube that included a fact table containing 6 dimensions, 30 key figures, and 100 000 000 rows. As parallel indexing is a common use case, we measured the indexing speed per thread in MB/s. The tests used five threads and were carried out with one indexing client and multiple indexing clients. We ensured that each host received a portion of data to index by splitting the index into 12 equal parts and assigned 2 parts per hosts. The aggregated throughput while indexing on central storage with one client was 106 MB/s and with six clients 71 MB/s, whereas on the PL it was 57 MB/s with one and 78 MB/s with six clients. In the single-client scenario, the low performance of the PL is explained by the pipelining of the data over the commodity 1 Gbps network infrastructure in contrast to the 2 Gbps FC of the CS. The throughput of the PL improves as the number of clients increases, and is limited by how much data can be sent through the network at once. The indexing throughput in the CS scenario seems to be limited by the need for synchronization in the cluster file system.

In summary, our PL does not degrade the query performance of the BWA. The indexing throughput of the PL is comparable to the current SAN solution when multiple clients are used. The PL read and write performance satisfies the BWA requirements.

6. CONCLUSION

The storage requirements of BWA are currently handled by expensive central SAN. The local hard drives on the BWA blades are used only for operating system and cluster administration software. In the current BWA setup, the further capacity of the local drives remains unused.

We propose a distributed persistence on commodity hardware especially for small and midsize customers. The PL is deeply embedded in the BWA, takes care of efficient data distribution over local hard disks, and provides reliability by software-based fault tolerance. We developed the solution by refining and adapting the standard principles of distributed file systems.

As our evaluation of the PL showed and our first prototype confirmed, the new approach allows BWA to be set up with a cost saving of about 50% yet still deliver good performance. With a small appliance, the potential cost reduction is more than 70%. This makes the BWA attractive to customers in the small and midsize business sector. More widely, our work suggests that adopting the proposed persistence model in an OLAP appliance can offer significant TCO reduction for appliance customers.

There are plenty more opportunities for improvement, which makes this approach even more compelling. The use of statistics for data replacement and replication, or cluster rebalancing following appliance changes, are just some of them. Introducing transactional behavior is another possibility.

REFERENCES

Adya, A. et al., 2002. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. *Proc. 5th Symposium on Operating Systems Design and Implementation*. San Francisco, USA, pp. 1–14.

Braam, P. J. and Schwan, P., 2002. Lustre: The intergalactic file system. *Proc. Ottawa Linux Symposium*. Ottawa, Ontario Canada, pp. 50–54.

Bindel, D. et al., 2000. Oceanstore: An extremely wide-area storage system. *Technical Report of University of California at Berkeley*. Berkeley, USA.

Boncz, P. A. et al., 2008. Breaking the memory wall in MonetDB. *Communications ACM*. New York, USA, pp. 77–85.

Chang, F. et. al., 2006. Bigtable: a distributed storage system for structured data. *OSDI '06: Proc. 7th Symposium on Operating Systems Design and Implementation*. Seattle, USA, pp. 205–218.

Dean, J. and Ghemawat, S., 2004. MapReduce: Simplified data processing on large clusters. *Proc. 6th Symposium on Operating System Design and Implementation*. San Francisco, USA, pp. 137–150.

Ferrin, B. and Plank, R., 2002. Total cost of ownership models: An exploratory study. *Journal of Supply Chain Management*. Volume 38-3, pp. 18–29.

Ghemawat, S. et al., 2003. The Google file system. *19th ACM Symposium on Operating Systems Principles*. New York, USA, pp. 29–43.

Hupfeld, F. et al., 2007. XtreemFS – a case for object-based storage in grid data management. *Proc. 33th Int. Conf. on Very Large Data Bases (VLDB) Workshops*. Vienna, Austria

Legler, T. et al., 2006. Data mining with the SAP NetWeaver BI Accelerator. *Proc. 32nd Int. Conf. on Very Large Data Bases (VLDB)*. Seoul, Korea, pp. 1059–1068.

Rhea, S. et al., 2003. Pond: The Oceanstore prototype. *Proc. 2nd USENIX Conf. on File and Storage Technologies*. Berkeley, USA, pp. 1–14.

Ross, J.A., 2009. SAP NetWeaver BI Accelerator. Galileo Press. Bonn, Germany.

Schmuck, F. et al., 2002. GPFS: A shared-disk file system for large computing clusters. *Proc. 1st USENIX Conf. on File and Storage Technologies (FAST)*. Monterey, CA, pp. 231–244.

Stonebraker, M. et al., 2005. C-store: a column-oriented DBMS. *Proc. 31st Int. Conf. on Very Large Data Bases (VLDB)*. Trondheim, Norway, pp. 553–564.

Stonebraker, M. et al., 2007. One size fits all? Part 2: Benchmarking studies. *Proc. CIDR*. Pacific Grove, USA, pp. 173–184.

Venner, J., 2009. *Pro Hadoop*. Apress, New York, USA.

Weil, S. A. et al., 2006. Ceph: A scalable, high-performance distributed file system. *Proc. 7th Symposium on Operating Systems Design and Implementation*. Seattle, USA, pp. 307–320.

Weil, S. A. et al., 2006. CRUSH: controlled, scalable, decentralized placement of replicated data. *Proc. 2006 ACM/IEEE Conf. on Supercomputing*. New York, USA.