DVTDS
Directory Server

LDAP Directory Benchmark
Cluster and Scaling Capabilities

A TeraCortex White Paper
June 2015

Table of Contents

Table of figures

Disclaimer

The material presented in this document has been worked with great care to correctness. However it is not a commitment for delivery of any kind nor does it imply any contractual binding in a legal sense.

1. Management Summary

About TeraCortex

For more than 20 years TeraCortex was active in IT consulting focused on development in large data base environments. Since 2012 we concentrate on the development of LDAP technology for subscriber data management in mobile and social networks. Further our products are targeted for scientific environments where large amounts of experimental data (particle accelerators, wind channels) must be stored and handled in shortest time.

About DVTDS

DVTDS stands for Distributed Virtual Transaction Directory Server, a new high performance standard LDAP server developed from scratch by TeraCortex. Beside its outstanding speed and exceptional scaling capabilities the server fully supports a set of cutting edge functionality like geo redundancy by multiple master replication, distributed transactions, triggers and view tables. For further details please refer to the DVTDS feature description [1].

About ELDC

ELDC is a free configurable high performance LDAP client supporting multiple parallel sessions. It is the reference implementation of the “Embedded LDIF for C” specifications. For details about ELDC please refer to [3]. The Embedded LDIF specifications are available as Internet Drafts at the IETF [4], [5], [6], [7] and at the TeraCortex web site. From there also the executable tool can be downloaded free of charge.

About this Benchmark

We executed a series of high volume benchmarks against a cluster of 16 virtual machines in the Amazon EC2 cloud. Each machine was equipped with our LDAP directory server DVTDS. The setup reached a throughput of **more than 22 million ACID compliant transactional random writes per second on a distributed data base with more than four billion entries.** To our knowledge this is by far the highest result ever reported for a distributed LDAP directory. Further our internet research about performance and scalability No SQL key value stores showed that DVTDS compares favorably against state of the art products like Aerospike, Redis and FoundationDB.

To relate this technical number to human experience: Imagine a social network with two billion subscribers, which is a substantial portion of the world's population. Assume that *all* of them are moving around at the same time and changing their position, any one reaching a new mobile cell every 100 seconds. Then this data base is able to track everybody's geographic location in real time. Doing so by use of a small set of virtual machines at a total hardware price of just US $40 per hour.

Or imagine a shot in a particle accelerator or other scientific experiment: The data base stores billions of key – value pairs at a rate of more than 40 million/s, having them ready for evaluation within minutes.

In addition to these outstanding results this benchmark shows:

- DVTDS linear performance scaling in cluster machine arrangements
- It's ability to run full ACID compliant distributed transactions across multiple data base nodes in clusters at full throughput
- It's ability to effectively handle large collections of key – value pairs
- It's extreme efficient bulk load interface which populates 17.2 billion entries in less then nine minutes
- Transactional and non – transactional write performance equal or better than read performance

For a DVTDS replication benchmark please refer to [12].

Here are the top level results:

	4.3 billion entries (Xeon Haswell)	**4.3 billion entries (Xeon Ivy Bridge)**	**17.2 billion entries (Xeon Ivy bridge)**
Data load and indexing	105 seconds	110 seconds	502 seconds
Object insert rate	41.3 Million / s	39.0 Million / s	34.2 Million / s
LDAP search request	22.0 Million / s	16.4 Million / s	5.6 Million / s
LDAP transaction request	22.0 Million / s	15.0 Million / s	4.9 Million / s
LDAP mixed search / modify (50% / 50%)	24.6 Million / s	17.5 Million / s	5.5 Million / s

From the very beginning DVTDS was designed with parallel hardware in mind. Its multi – threaded architecture is optimized to make maximum use of multiple cores, hard disks and memory channels. For this reason it scales excellently with modern many – core machinery. Moreover it is not restricted to a single instance deployment. Instead it fully supports distributed LDAP operations and transactions across multiple instances running on the same or different machines while still maintaining a single consistent logical data model from the client point of view. Throughput and the amount of data can be scaled to largest deployments by just adding more memory, hard disks and / or machines to the system. Unlike most well – established LDAP directories it is able to process highest update workloads in real time. Response times well below as 30 micro seconds are achievable. The server comes in two flavors:

- As in – memory data base without hard disk back end. This version is intended for highest volume traffic at moderate data volumes of up to several Terabytes, subject to the amount of available RAM. The cluster benchmarks published in this document ran on this type of server.
- As hard disk based, memory mapped version for increased storage requirements scaling into the Petabyte range, subject to the amount of available hard drives. Chapter 3.6 describes such a benchmark

2. Comparison with other Benchmark Reports

There is quite a number of benchmark reports for other products from various sources, in varying quality and executed on single machines or in cluster environments of different sizes. One could think to normalize them to a common metric like throughput per physical CPU core and GHz clock speed. Still such comparison can be doubted, because other factors influence the outcome, among them:

- Network speed in cluster environments
- Memory generation and speed
- Processor generation, architecture, speed and inter – CPU bandwidth
- On disk or in memory operation
- Number and speed of hard disks, magnetic or solid state drives
- Native or virtual hardware
- Amount of test data loaded

The last point is of particular interest. Today's machines usually have NUMA (Non unified memory architecture) along with a two socket main board and RAM attached to a particular processor. This means, that data needed by one processor may be located in the RAM of the other which requires a time consuming inter – CPU communication before the data can be processed. With small amounts of data in the benchmark setup this effect is not visible. But when raised to 50% or 80% of the available RAM the results change dramatically. Data bases used to deliver millions of operations per second suddenly drop by fifty or more percent. Benchmarks dealing with just 10 or 50 million small objects (or key value pairs) tend to hide these problems. This is why we chose the largest possible data set for our tests. DVTDS turns out faster in any case for In Memory operation. On Disk only Aerospike and Redis seem to be faster, but take care: Their read write ratio was 50 / 50 non transactional while the DVTDS test was write only transactions. Further their amount of data was less than 2% of the set used in DVTDS. On the other hand they had two internal SSD drives (r3.8xlarge) while DVTDS used eight (i2.8xlarge).

By the way Oracle achieved a LDAP world record when they delivered the shown result in 2013 at the launch of the T5-2 Sparc server. One year later DVTDS broke it with a simple desktop PC at 3% of Oracle's hardware price.

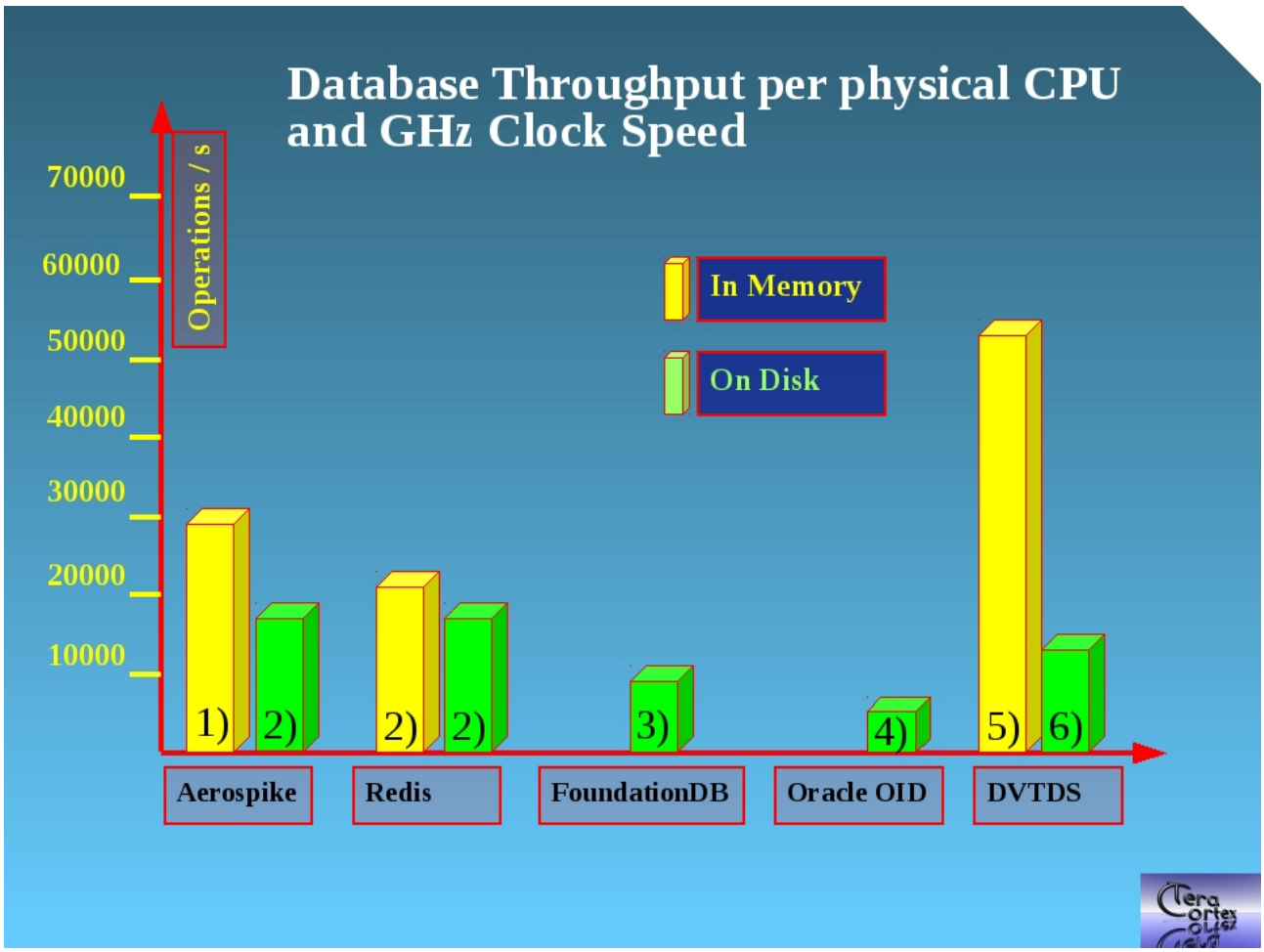

Figure 1: Throughput comparison from various benchmark reports

Remarks

1) Aerospike running on single native two socket server 2 x Intel E5 2699v3, 50 million records, read – write 95 / 5. [9]
2) Aerospike and Redis running on single Amazon r3.8xlarge virtual machine, 10 million records, read – write 50 / 50. [10]
3) Foundation DB running on Amazon cluster of 32 x c3.8xlarge virtual machines, 33 million records per machine, write only. [11]
4) Oracle running on native two socket sparc server T5-2, 50 million records per machine, read only. [8]
5) DVTDS running on single Amazon c4.8xlarge virtual machine, 256 million records per machine, read – write 50 / 50
6) DVTDS running on single Amazon i2.8xlarge virtual machine, 256 million records, write only

3. Benchmark Setup

3.1 Hardware and Operating Systems

The following machines participated in the benchmark:

	4.3 billion entries	**4.3 / 17.2 billion entries**
Machine Type	Amazon EC2 c4.8xlarge, 36 virtual cores	Amazon EC2 r3.8xlarge, 32 virtual cores
Number of machines	16	16
CPU	Intel Xeon E5-2666 v3 (Haswell) @ 2.3 GHz	Intel Xeon E5-2670 v2 (Ivy Bridge) @ 2.5 GHz
Memory	60 Gbyte DDR4 @ 2133Mhz per machine	240 Gbyte DDR3 @1600 MHz per machine
Storage	In memory	In memory
Network	10 Gbit/s ethernet, same sub net placement	10 Gbit/s ethernet, same sub net placement
Operating system	SLES 12 / 64 Bit	SLES 12 / 64 Bit
Directory server	DVTDS 3.2 / 64 Bit In memory	DVTDS 3.2 / 64 Bit In memory
Client	ELDC 1.005 running on each machine	ELDC 1.005 running on each machine
Client – Server connection	LDAP / TCP via local host network interface	LDAP / TCP via local host network interface
Server – Server connection	LDAP / TCP via 10 Gbit ethernet, same sub net placement	LDAP / TCP via 10 Gbit ethernet, same sub net placement

3.2 Populating the Data Base

In preparation of the tests we developed Shell scripts and ELDC sources which created the entire data base content by a binary sub division algorithm. Like this the loader client data arrived at the DVTDS data base servers already in indexed form. For this reason the server was more or less relieved from the task of index creation. This technique ensured fastest possible population of the data base. From the physical point of view the data was partitioned into 64 shards, where each of the 16 machines took 4 shards.

Loading was performed by use of the built – in parallel bulk load facility. This function of DVTDS is able to read multiple streams of BER encoded LDAP add operations directly from local files or FIFO devices. Indexing was accomplished on the – fly during the load process. The only indexed attribute was the naming attribute “uid”. We used ELDC to generate the BER encoded streams from a template. It then fed the parallel streams into local FIFO devices while DVTDS was sitting at the FIFO outlets, reading the streams and converting the data to internal representation. This technique avoided the time and disk space consuming intermediate storage of load data. The entire data set was stored in main memory. Per shard four parallel BER encoded streams were used, which amounted to a total of 16 loader sessions per machine. The same process ran at the same time on each machine with different data content. The picture below shows the relevant components for a single machine.

Please note, that this power of two setup was only chosen for data population efficiency. DVTDS supports any number of keys and objects and any type of keys, including non – numerical ones like names, e-mail addresses or whatever.

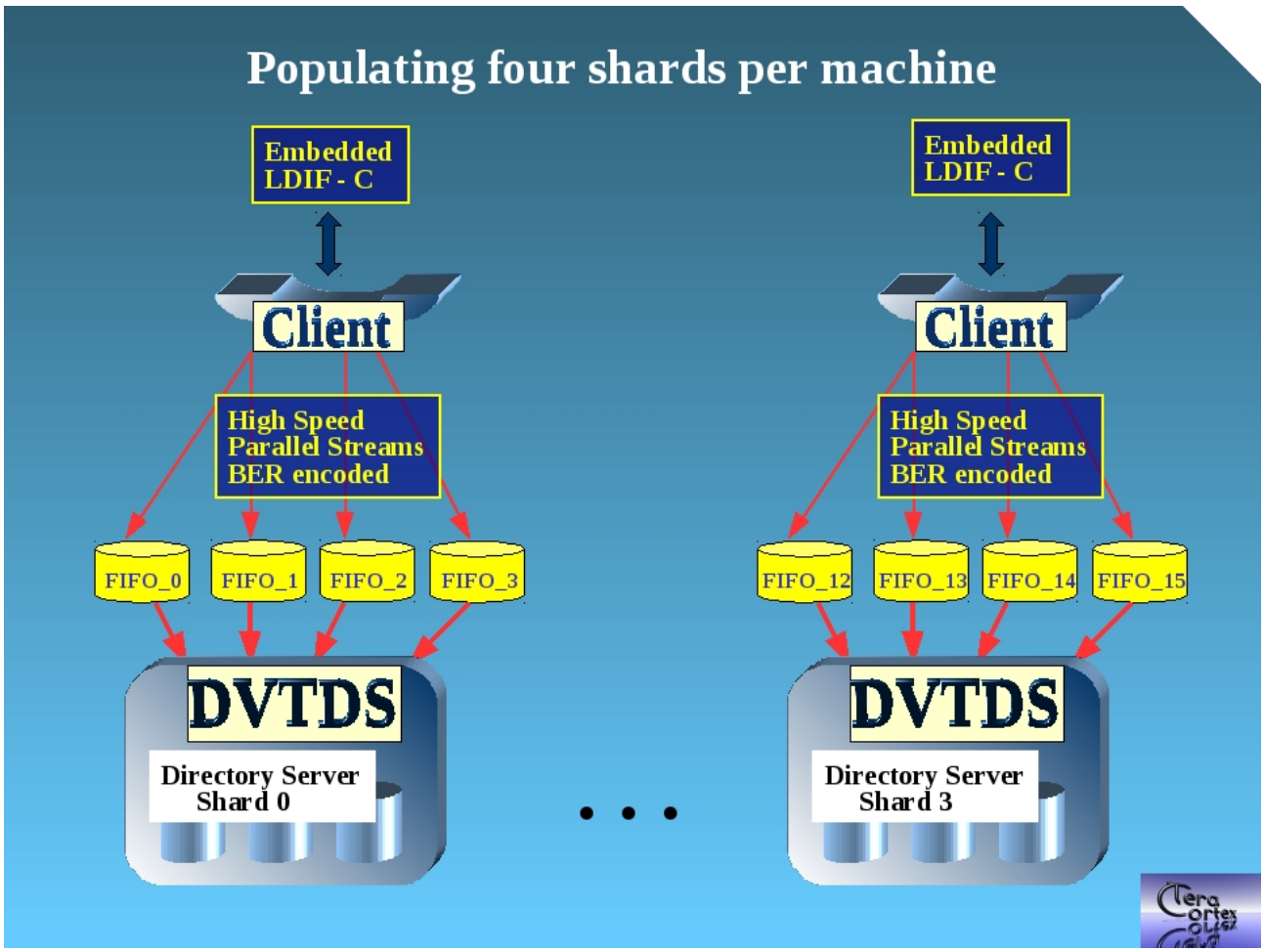

Figure 2: Data population mechanism

3.3 The Data Model

The logical data model consists of a flat structure below the root distinguished name dc=com". A LDAP client sees all the billions of entries below this root. Each entry holds just a single attribute. This arrangement was chosen for easy comparison with other LDAP benchmark reports and No SQL key value stores. Please note that DVTDS is not restricted to such simple data models. It supports multiple values per attribute, multiple attributes per entry, multiple entries arranged in tree like structures and multiple top level tree roots. In fact in a real world deployment a subscriber or other type of business subject (experimental data) would almost always consists of a more or less complex sub tree of objects, attributes and values and the root of each subscriber's sub tree carries one or more of its identities that are used as access keys. For the sake of simplicity the initial attribute values were all the same across the entire data set. The picture below shows the logical data model:

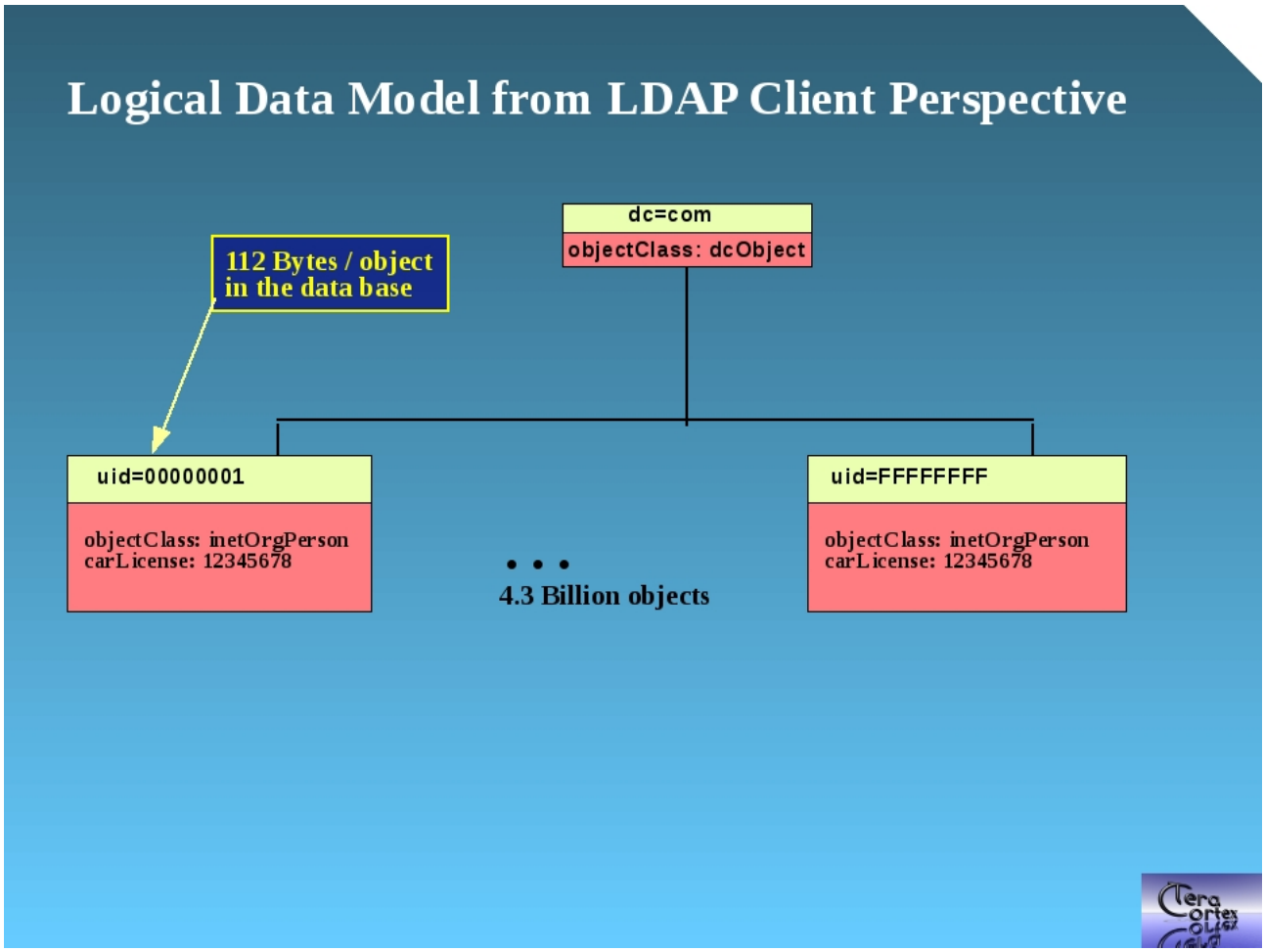

Figure 3: Logical data model (client perspective)

Each object is member of the object class inetOrgPerson, which is one of the LDAP standard model object classes. It holds the single attribute carLicense with an initial 8 byte value, making up for 112 Bytes of storage space per object. There were no operational attributes generated by DVTDS. The server supports single root data models (the one we used in the benchmark) as well as multiple roots (or naming contexts). From the client point of view the former type of model displays a single logical object space which is preferable in most situations. In the case of multiple roots the client sees multiple or partitioned object spaces. Other LDAP products enforce partitioned data models if multiple hard drives are used for storage. DVTDS has no such restriction because it implements a strict separation between the logical appearance of the data and its physical distribution over the underlying hardware. From the perspective of LDAP clients the underlying physical arrangement is not visible. They just see a homogenous directory information tree as if connected to a single LDAP serve instance.

From the physical point of view the data is partitioned into 64 shards, where each shard holds a unique subset of keys and keys are disjoint across all shards (almost no data redundancy). In fact, “almost no redundancy” means the following: There is a small set of top level key objects (127 for a 4.3 billion object data base) that are present in either shard. Each top level key object represents a range of keys on a specific shard. They do not contain any attribute values. Instead each one carries a reference to a shard where objects falling in this range can be found. Requests hitting such a reference are automatically forwarded to the referenced target.

We chose this binary sub division model only for economic reasons: It guaranteed the fastest possible data population, thus kept our Amazon EC2 charges to a minimum. DVTDS is not restricted to such regular arrays (tables) of objects. It supports any number of objects in arbitrary tree structures, aliases (short cut references) pointing across local data content, remote references pointing across different server nodes and view tables as known from relational data bases. In any case the clients need not care about of this server – side physical machinery because it is handled internally by DVTDS. The image below depicts the physical data arrangement of this benchmark.

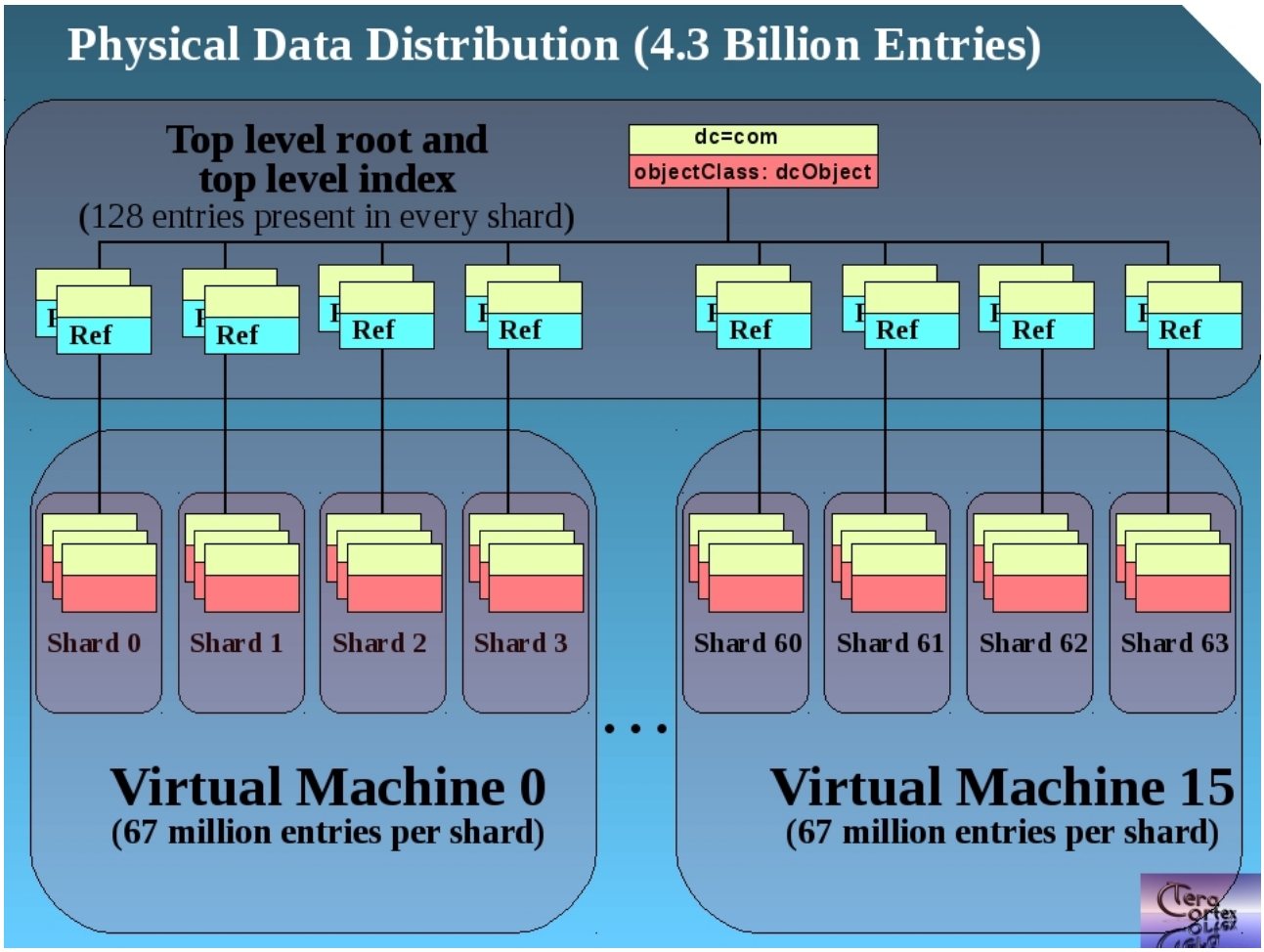

Figure 4: Simplified physical data model (server perspective)

3.4 Benchmark Scenario

All benchmarks were executed in the following manner:

- On each machine either of four client processes starts 1 … n parallel sessions by connecting via TCP/IP to the corresponding shard on the same machine
- For each connection the client sends a simple bind request, thus establishing one ore more LDAP sessions. The server associates the bind credentials with an access control regime and keeps to it for every request in the session
- After session initiation the client sends a series of requests or transactions (search, modify) to the server and receives the responses
- All test cases use asynchronous mode. This means that clients bundle a series of requests into one or more TCP packets and transmit them in burst mode. The server processes all requests in the bundle before sending the corresponding bundle of responses. In case of non transactional requests the server then waits for the next bundle to come in. In case of transactions it waits for the final commit or rollback directive from the client. The length of the asynchronous queue (number of requests in a bundle or length of a transaction) is 1000.
- Each request targets a single entry by its distinguished name (the key).
- Modify request in mixed mode or transaction mode change the value of the attribute. The value lengths varied between 8 bytes and 100 bytes dependent on the random keys. In average they are 50 bytes long.
- Distinguished names are chosen by random from an array of 100000 random values ranging across the entire key set. With each invocation of a client process a new random set is generated. From a statistical point of view all entries have an equal chance to be targeted. Based on this general outline the random distribution is shaped in a way that ensures that each request bundle hits always three different shards. This enforces in any case true distributed operation for all requests and transactions.
- Each single benchmark is terminated by the client by sending an unbind request for each established LDAP session

- After having terminated all sessions each client process calculates the throughput by simple division of the number of requests through the elapsed time
- The control script runs in a loop over all 64 shards. First just one client is invoked. Then two clients are invoked connection to the first and second shard and terminated after having delivered their results and so on. Although there is a 1:1 direct connection between a client process and a shard, the set of random keys used by each client forces each shard to propagate 60% of the requests to two other shards. Indirectly each client hits three shards

4. Benchmark Execution

4.1 General Approach

We performed all tests by repeated execution of scripts from the Linux command line of the control work station. The latter connected via SSH to each machine the Amazon EC2 cluster. With each invocation we increased the number of affected shards. Further we varied the number of parallel LDAP sessions fired by the test client and we varied the operation types (read, read mixed with non transactional write, transactional writes). As can be expected from parallel implementations the throughput increased with the number of parallel sessions. Further it increased with the asynchronous queue length. The reason is quite simple: Most LDAP request messages and LDAP response messages are much smaller than the TCP MTU (maximum transfer unit, 1500 bytes on many systems). Using asynchronous operation tends to better fill the available TCP packet size, thus making maximum use of the underlying network resources. This technique also avoids response time issues associated with the TCP Nagle algorithm. DVTDS and ELDC support the LDAP queue length control that enables the client to tell the server the preferred length of the asynchronous queue. This leads to a two – sided agreement about the optimum network utilization. The LDAP queue length control specification is available as Internet Draft at the IETF and at the TeraCortex web site.

4.2 Common Observations

In the tests we observed the following effects:

- The data base servers ran stable all the time even under highest pressure
- The distributed data base showed perfect linear scaling with the number of client sessions and affected shards as long as not bottle – necked by machine resource constraints
- Increasing the total number of client sessions above the total number of virtual cores in the cluster resulted in performance degradation
- The client CPU consumption was about 20% of the server CPU consumption. As they ran on the same machines as the servers we expect an additional performance win if clients would run on their own hardware
- Writes were faster or as fast as reads
- Performance increased with the length of asynchronous queues / number of requests per transaction. Queue lengths between 50 and 200 seem reasonable. Increasing the queue length above 200 (as we did) gives smaller and smaller advantages

4.3 Read operation (LDAP Search)

The below diagram shows the LDAP search operation throughput over the number of client processes working against the cluster data bases. Please note, that each client process started up to eight independent worker threads, At maximum 512 parallel client sessions were active, leading to as many primary server sessions. Due to random key distribution every client session forced the servers to propagate 60% of the requests to subordinate DVTDS nodes, invoking secondary server sessions in the subordinates. The whole arrangement led to parallel execution of more than 1500 server sessions. The graph also shows that Amazon r3.8xlarge instances (Intel Ivy Bridge architecture) with six threads are on par with c4.8xlarge instances (Intel Haswell architecture) using four threads. This seems consistent with internet sources [9] reporting a 56% performance advantage of Xeon Haswell over Ivy Bridge. At the high end first signs of machine saturation are visible.

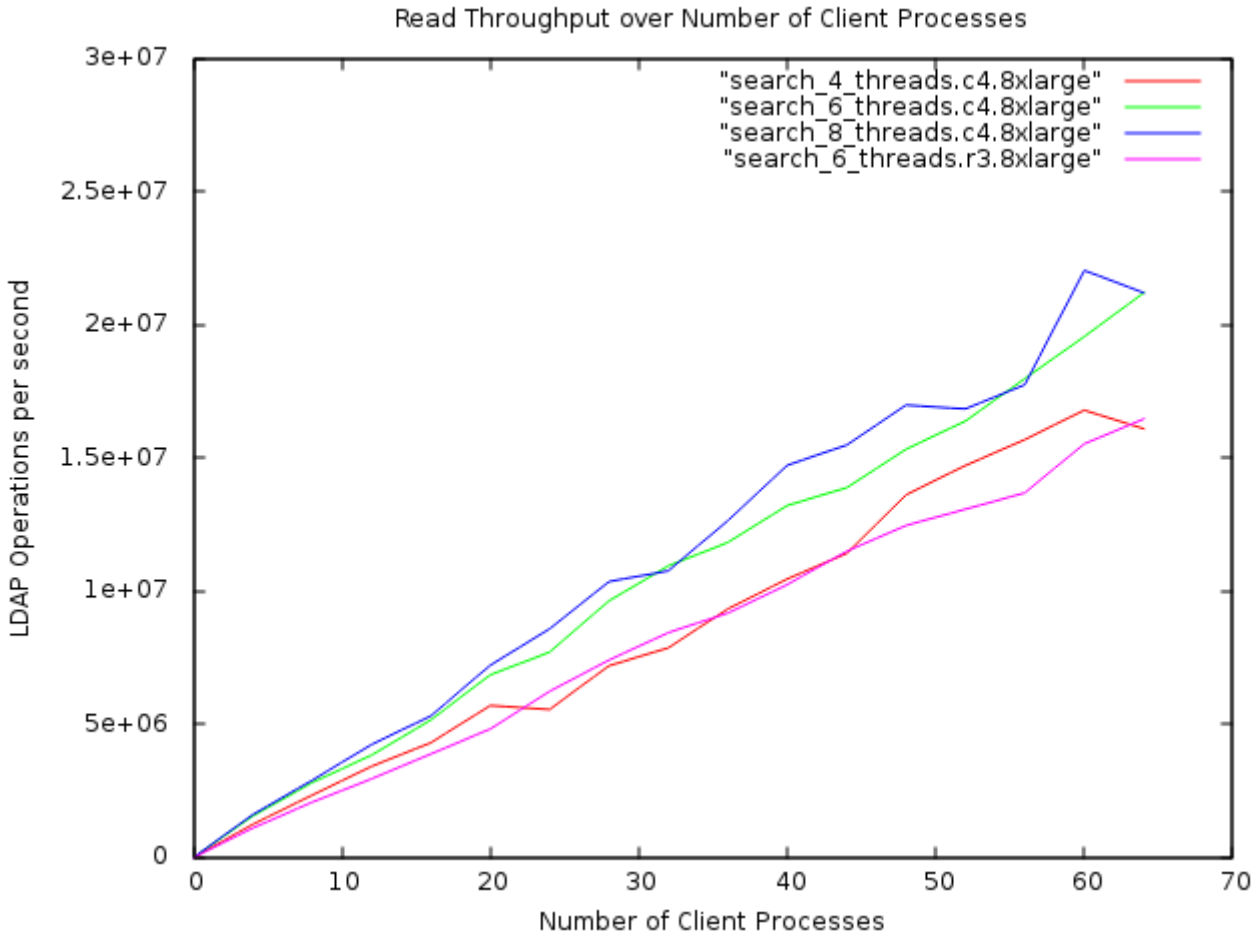

Figure 5: Throughput diagram for search requests

4.4 Read / Write mixed operation (LDAP Search / Modify)

See below the scaling diagram for mixed mode operations (50 / 50 LDAP search / modify). Both request types ran in non transactional mode in this test. The metrics are similar as for pure searches but up to 9 worker threads were used for each client process. Same as for pure search the throughput scales in linear fashion, but shows no signs of saturation with over 1700 parallel sessions present in the distributed data base. On c4.8xlarge instances the test was executed with 8 and 9 threads per client process and on r3.8xlarge instances we used 6 threads per client process.

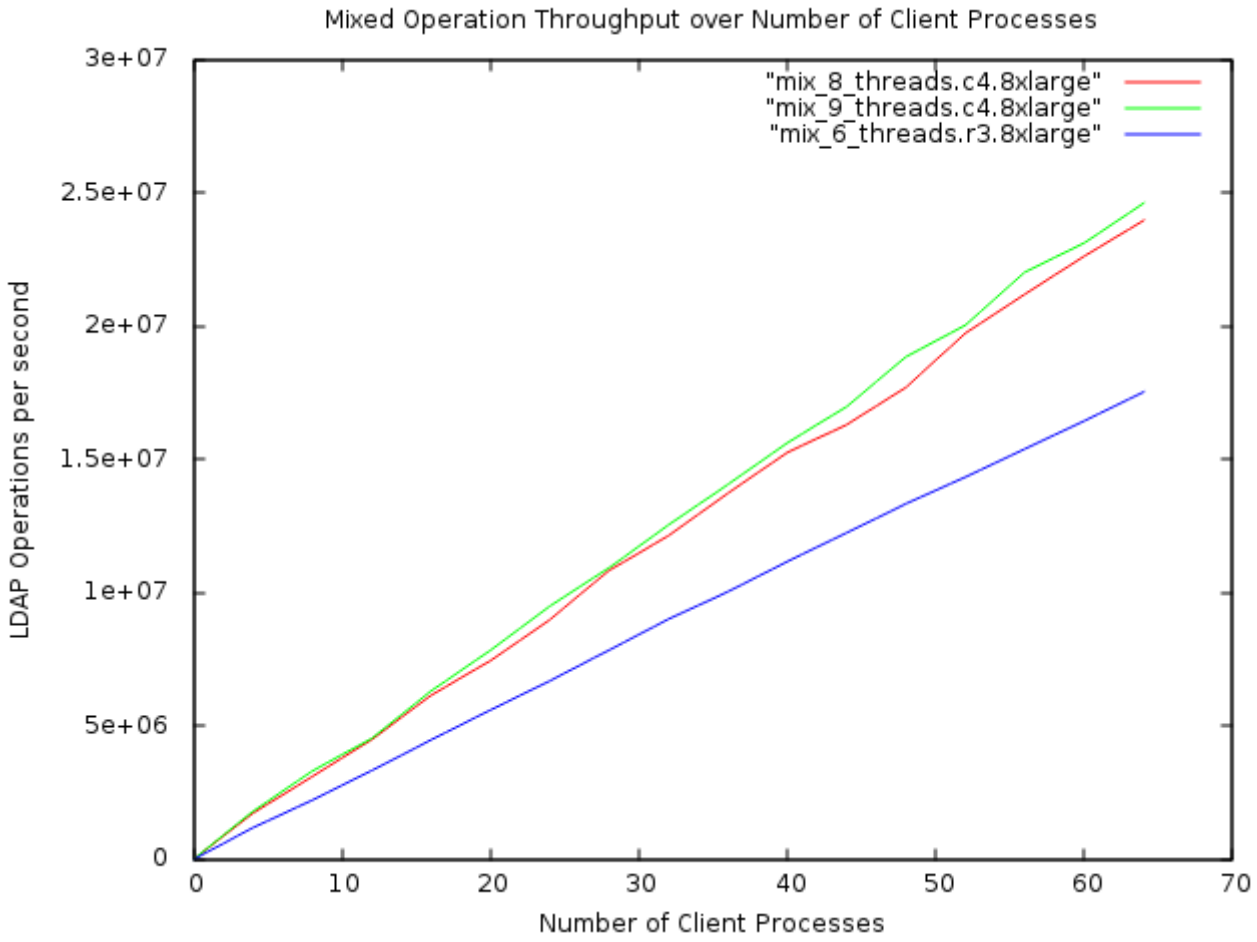

Figure 6: Throughput diagram for mixed read / write requests

4.5 Write transactions (Transactional LDAP Modify)

In this test we used full ACID compliant LDAP transactions. Due to the random key distribution almost any transaction was split internally by DVTDS and the parts of the transactions affecting remote DVTDS nodes were propagated across the network to their appropriate targets. The particular responses were collected and transmitted to the clients which were not aware at all that a distributed operation had taken place. The ability to hide the complex physical data distribution from the clients while still delivering transaction consistency at unmatched level of performance is an outstanding feature of DVTDS. In transaction mode first signs of machine saturation are visible at highest throughput.

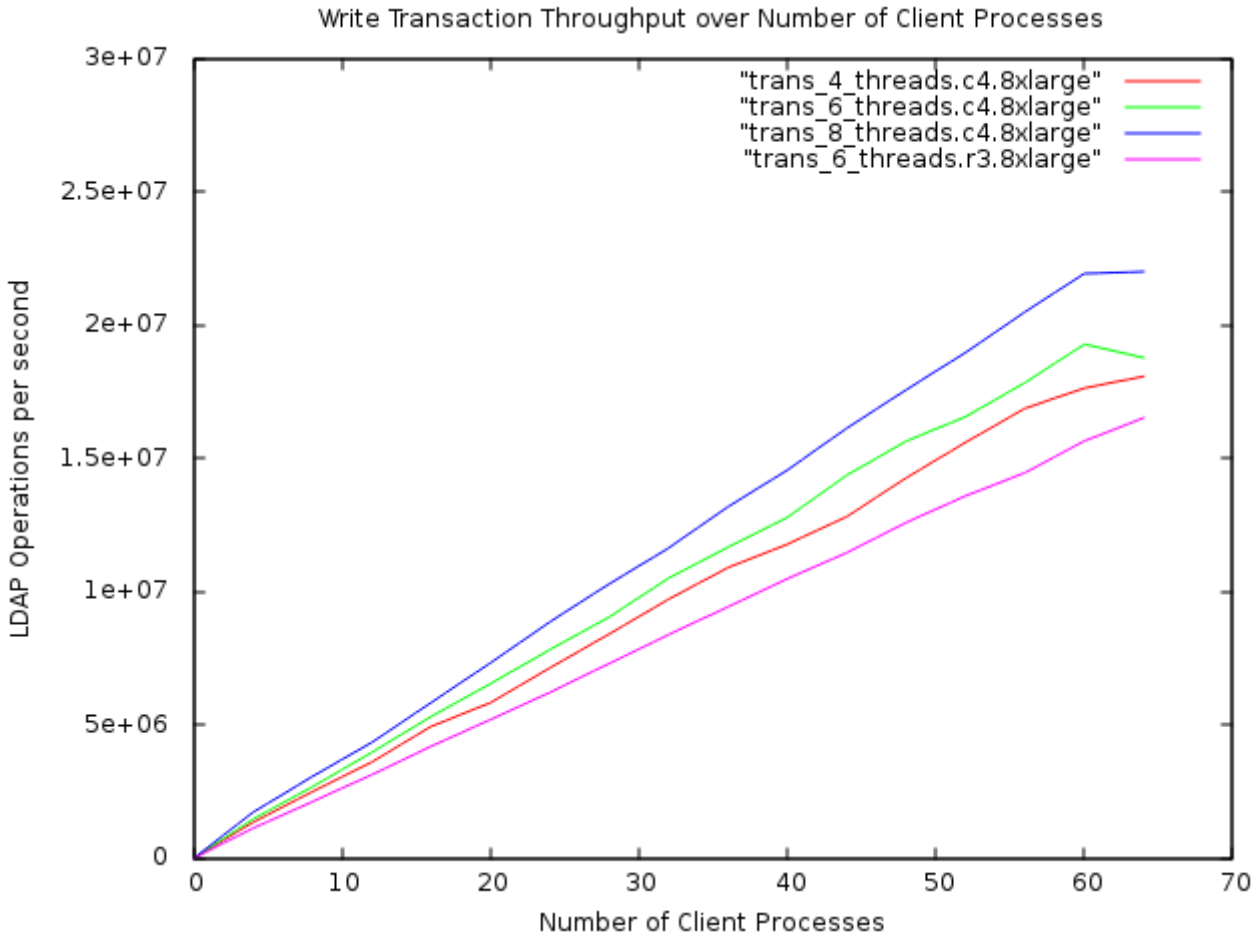

Figure 7: Throughput diagram for write transaction requests

4.6 In Memory versus On Disk Operation

DVTDS supports immediate persistence by memory mapped files. These files may be raw (block) devices, logical volumes under control of a volume manager or cooked space inside a file system. This feature adds a high level of reliability to the data base, especially when combined with mirrored disks. However, it comes at the price of a performance penalty. The latter can be mitigated to a large extend by help of multiple disks because DVTDS' internal multi threaded design is optimized to make maximum use of parallel hardware at CPU, RAM and disk level. To put this into numbers we executed a comparative benchmark between the In Memory and the On Disk version of DVTDS. All operations were executed on the same machine, with same client programs and DVTDS tuning configurations, once in memory, once on disk. Here is the specification of the setup:

	256 Million entries
Machine Type	Amazon EC2 i2.8xlarge, 32 virtual cores
Number of machines	1
CPU	Intel Xeon E5-2670 v2 (Ivy Bridge) @ 2.5 GHz
Memory	240 Gbyte DDR3 @ 1600 MHz per machine
Storage	On Disk, 8 x 800 GB SSD, internal, direct raw device, no volume manager
Network	Localhost loopback device
Operating system	SLES 12 / 64 Bit
Directory server	DVTDS 3.2 / 64 Bit On disk high yield
Client	ELDC 1.005 running on each machine
Client – Server connection	LDAP / TCP via local host network interface
Server – Server connection	LDAP / TCP via local host network interface
Queue length / Transaction length	1000
Client threads per shard	6
Shards	4
Subscribers	256 Million

This is the same type of hardware as the r3.8xlarge instances used for our in memory cluster benchmark. The only difference is the presence of eight internal 800 GB SSD drives. We could also have used Amazon's network attached EBS drives, but configuring them to acceptable performance is either cumbersome or expensive because Amazon enforces a disk quota system on them that is bound to the size of the drives. As a consequence we had to rent five hundred times the needed storage capacity to get superior IO performance. This is why we preferred the probably slower internal drives.

The benchmark was organized like a stripped down version of the previous cluster benchmark. Now the cluster consisted of just four shards sitting on the same machine. Random keys were configured to hit all shards with equal probability which enforced distributed transactions across the four DVTDS nodes. Each shard was configured to use two of the eight internal SSD drives.

Here are the top level results:

	256 Million entries (On Disk)	**256 Million entries (In Memory)**
Data load and indexing	290 seconds	98 seconds
Object insert rate	0.93 Million / s	2.73 Million / s
LDAP search request	1.05 Million / s	1.19 Million / s
LDAP transaction request	0.64 Million / s	1.15 Million / s

We ran the benchmarks over four minutes to discover situations were updates are synchronized to the disk hardware or pages are fetched from there. The benchmark parameters which influence performance scaling (Number of threads, request queue length, transaction length) were kept at fixed values. The following graph visualizes the results for search operations. The green lines represents the in memory results, the red line stands for on disk operation. One can clearly see the periodic physical disk access were throughput drops by roughly 20%. However, in average the On Disk mode amounts to more than 70% of the In Memory operation which underlines the efficient implementation.

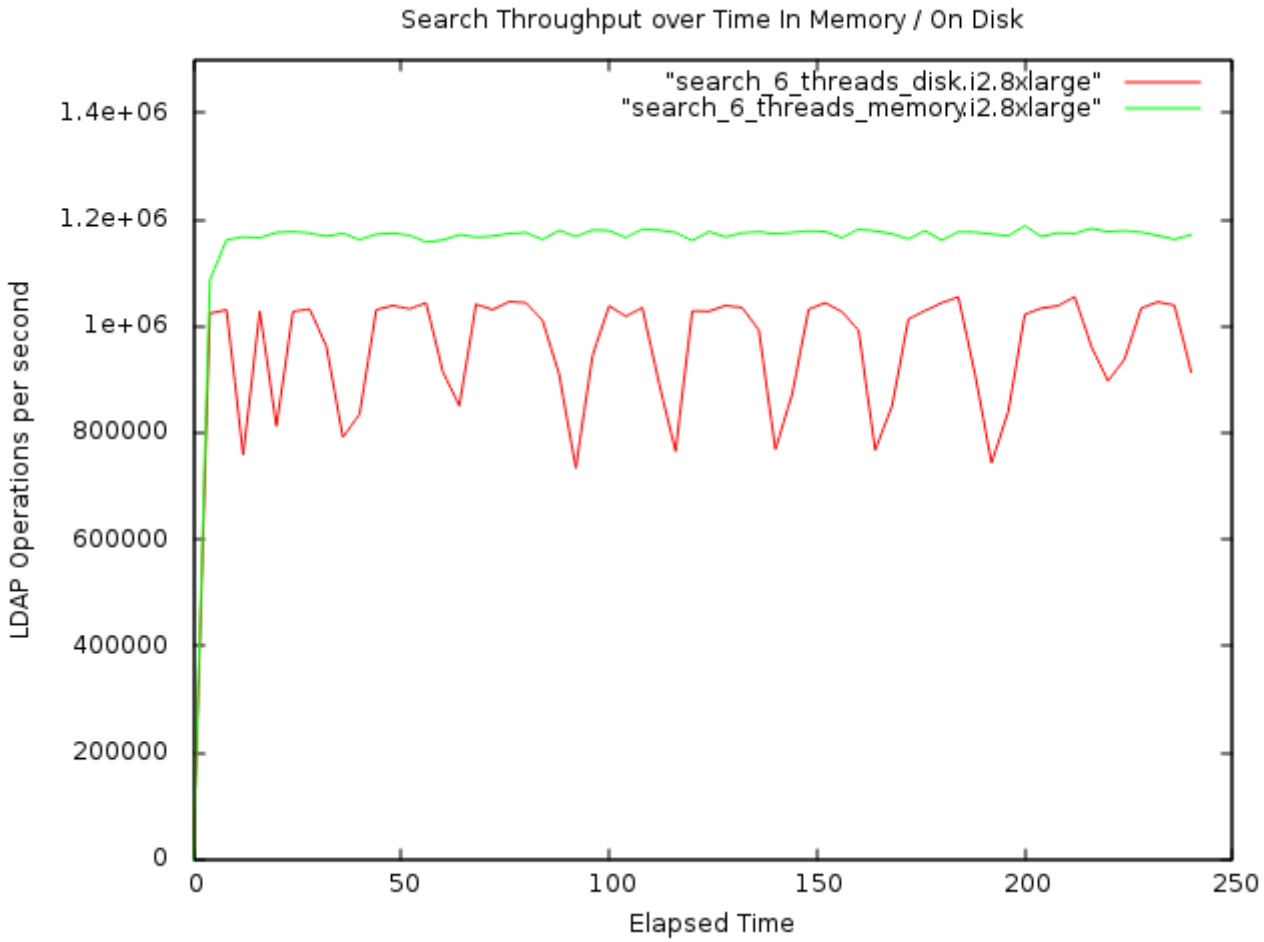

Figure 8: Throughput diagram search requests in memory versus on disk

The following picture shows the results for write transactions. Again the green line displays the In Memory case, the red line stands for On Disk:

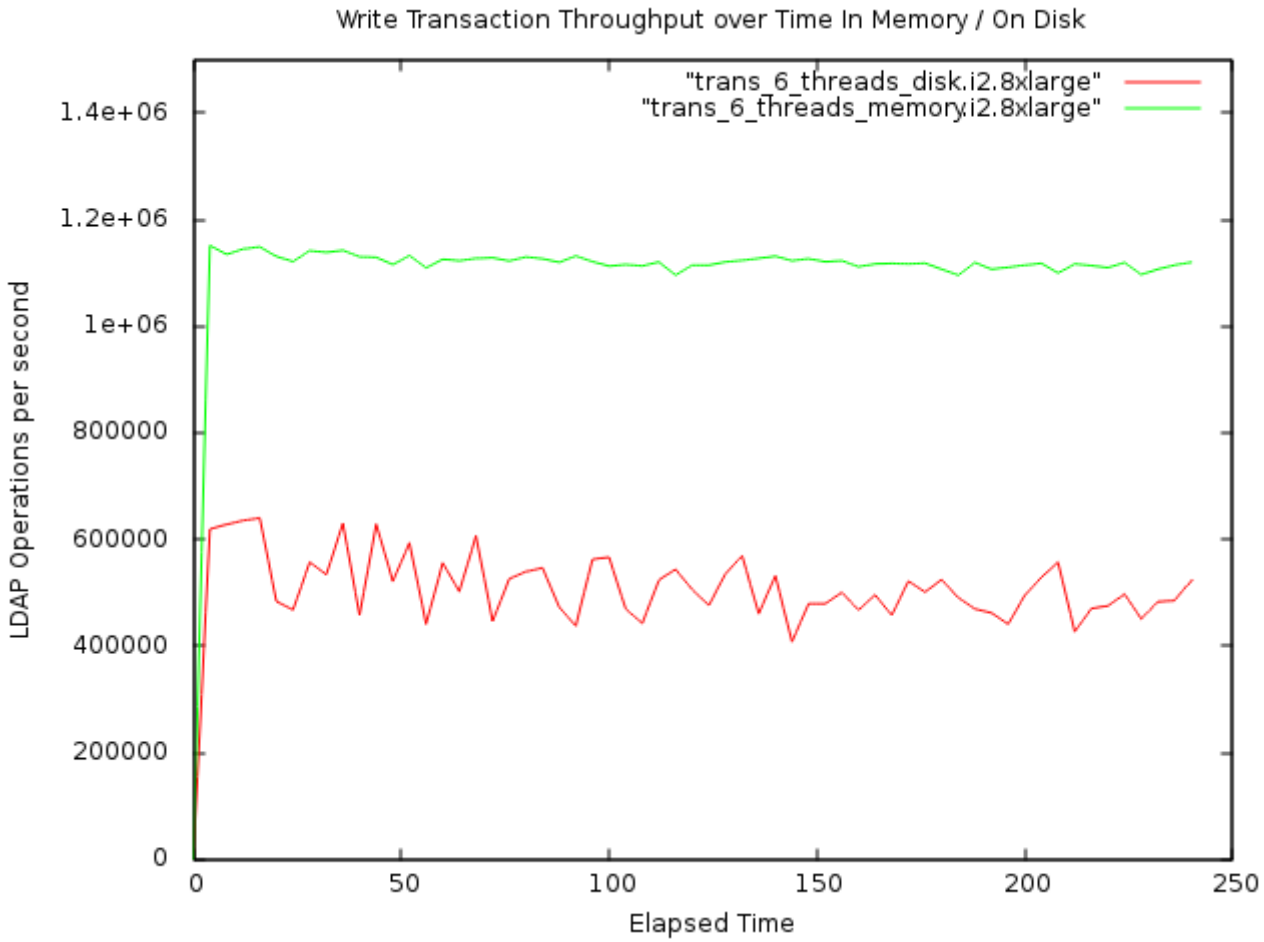

Figure 9: Throughput diagram write transactions in memory versus on disk

In our experience the principal behavior of write requests On Disk is always similar: First the performance is quite high but when synchronization to physical drives begins the performance drops to the average that the hard disk sub system is able to sustain. For this benchmark twice the number of hard drives would enable DVTDS to catch up to the In Memory performance.

4.7 Single Machine Results

This test follows the same setup as in the previous chapter but is aimed at the scalability of

DVTDS on single c4.8xlarge instances. We installed a distributed data base with four DVTDS nodes (shards) on the machine and used a random key distribution that forced any request bundle / transaction to be split across two shards. The table below summarizes the setup.

	256 Million entries
Machine Type	Amazon EC2 i2.8xlarge, 32 virtual cores
Number of machines	1
CPU	Intel Xeon E5-2666 v3 (Haswell) @ 2.3 GHz
Memory	60 Gbyte DDR4 @ 2133Mhz per machine
Storage	In memory
Network	Localhost loopback device
Operating system	SLES 12 / 64 Bit
Directory server	DVTDS 3.2 / 64 Bit On disk high yield
Client	ELDC 1.005 running on each machine
Client – Server connection	LDAP / TCP via local host network interface
Server – Server connection	LDAP / TCP via local host network interface
Queue length / Transaction length	1000
Client threads per shard	1 – 9
Shards	4
Subscribers	256 Million

The picture shows the scaling throughput for read (search), read / write (search / modify, 50 / 50) and transactional writes (modify). Please note, that each shard was targeted by one client process with up to 9 threads. At maximum 36 client sessions were active in the distributed data base.

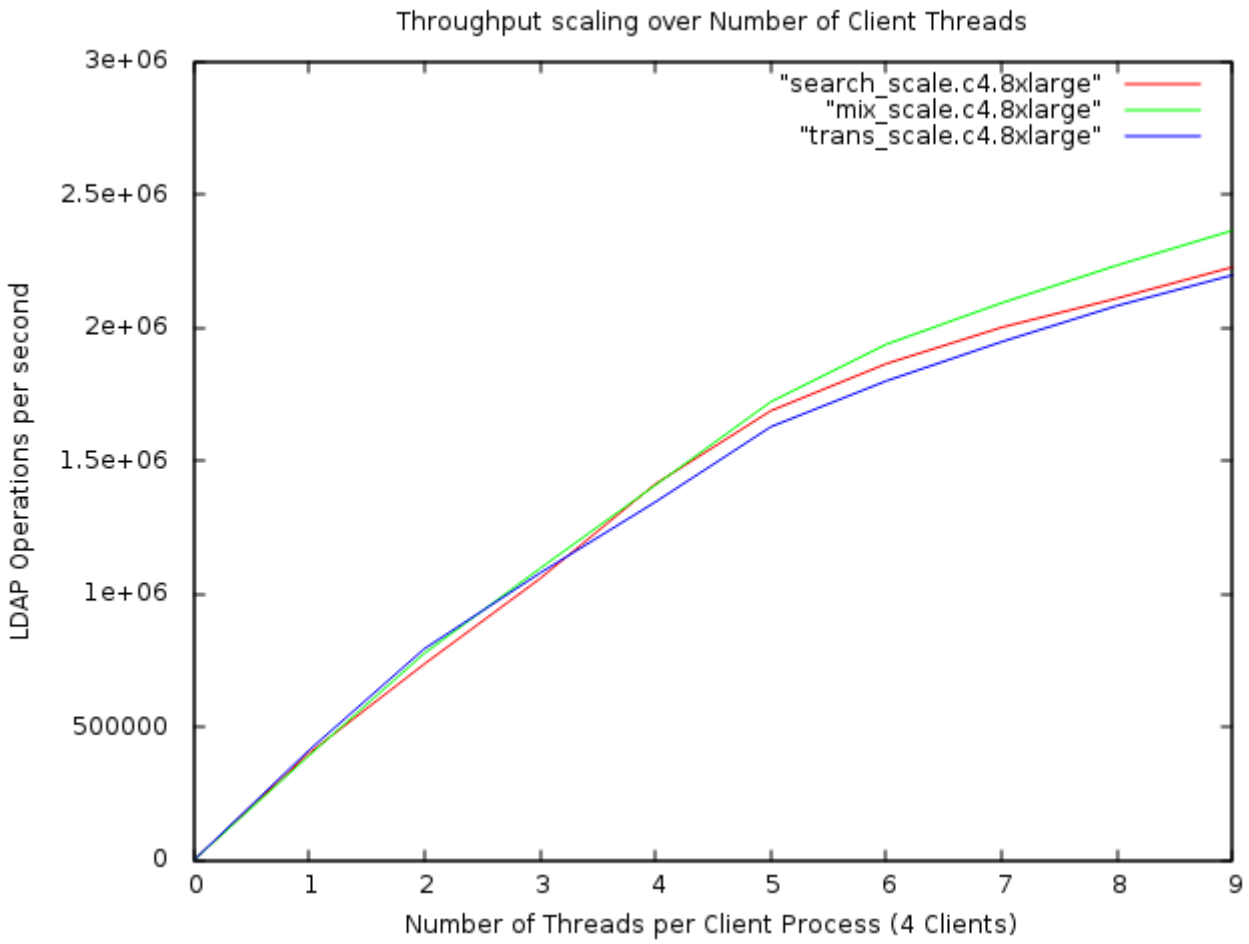

Figure 10: Throughput diagram for a single machine

5. Transaction Semantics

DVTDS offers full compliance with international standard LDAP transactions according to RFC5805. As this specification makes no statement about distributed transactions, DVTDS implements them as compatible extension. The specification uses an explicit transaction begin operation. The following sequence diagram shows the principal process. The client initiates the transaction with a transaction begin request. It receives from the front end server a transaction begin response which also contains a transaction identifier. Then the client sends a number of update requests where each request carries the transaction identifier as part of an appended control. The front end server does the same in direction of any subordinate servers, means: towards them it behaves like a transaction client (yellow arrows at the top of the diagram).

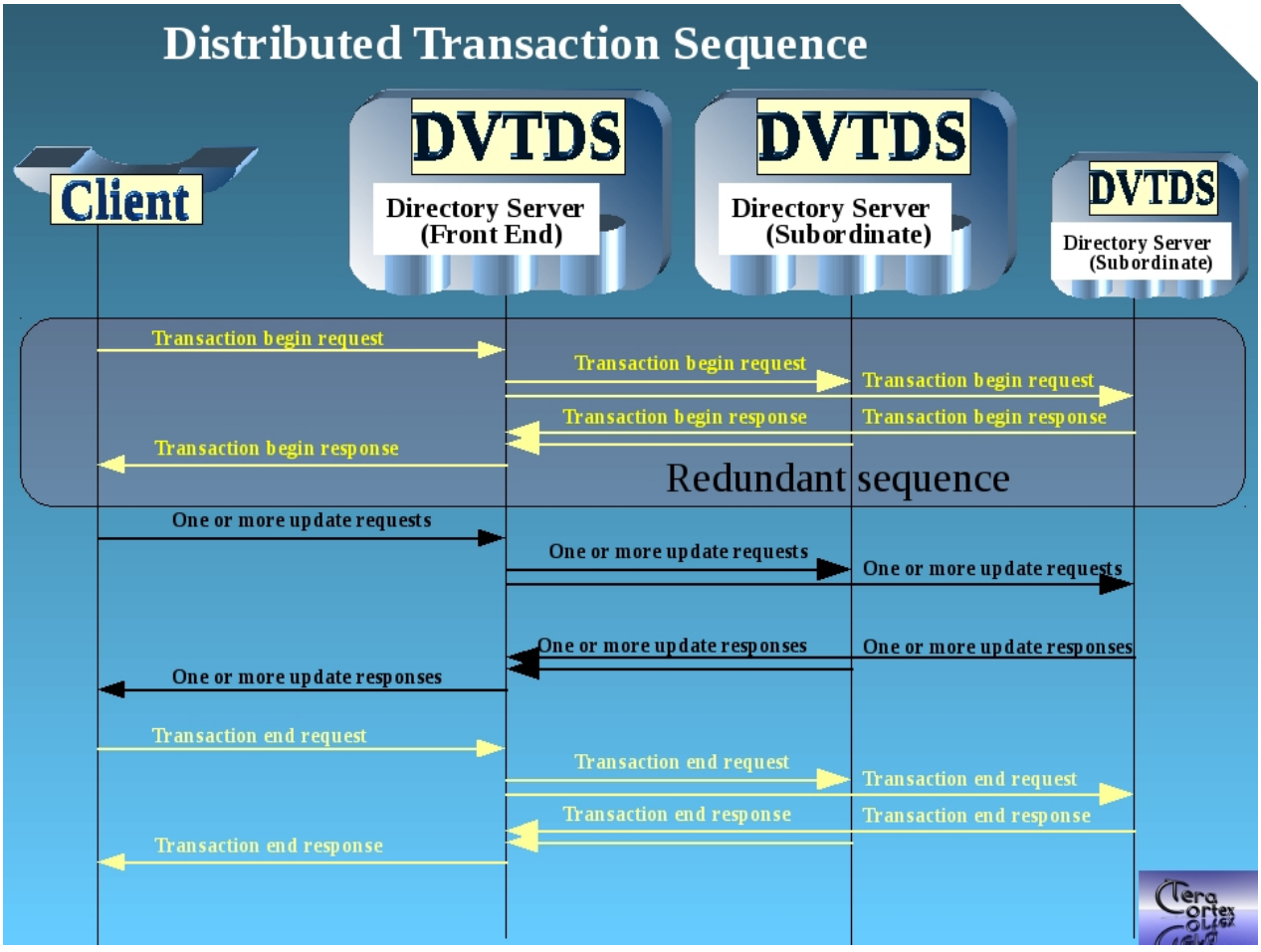

Figure 11: Transaction sequence diagram

But it has to perform one more step: The collection of requests may partially target data content locally stored in the front end and partially target data content stored in one or more subordinate servers. The precise data distribution depends on the keys used by the client to access the target objects. So the front end server must split and distribute the requests and collect the responses from the subordinate servers. When all responses have arrived, it must merge them with the results from its local operations and transmit them all together to the client. (Black arrows in the middle of the sequence diagram).

It is finally the client who decides upon the received responses whether to commit or rollback the whole transaction. This is done in the final sequence (yellow arrows at the bottom of the diagram).

Now, whats wrong with the rounded rectangle tagged as “redundant”? RFC 5805 states that each transactional request has to append a LDAP control carrying the transaction identifier. But why should the client not chose one by it self? If the TID is only used in a particular client session and the server has a clear separation between different client sessions, there is no danger that different transactions get puzzled. In this case the server may detect a transaction begin with the appearance of the first appended transaction control carrying an identifier on behalf of the client. This optimization does not compromise the ACID paradigm but avoids the (by its nature) synchronous and time consuming process of establishing the transaction context across all parties.

Still there is one more redundancy to be tossed out which is not apparent from the sequence diagram. Inside a transaction there are typically no or very few operations that do not belong to it and may be handled by the server separate from it. But RFC 5805 asks for the presence of a transaction control for every update. The server knows when it is inside a transaction and needs no reminder for this fact. Once inside a transaction it can in principle safely assume a transactional update if the control is *absent, unless a non – transaction control accompanies an operation* that may be handled outside of the transaction. This technique avoids transmission of the same information again and again over the network. Instead of flagging transactional membership for the vast majority of requests it flags non – transactional membership for the small minority.

In extension to RFC 5805 DVTDS supports both optimizations by online activation on behalf of the data base administrator. They were used throughout this benchmark and accelerated all distributed transactions almost to the point of non transactional updates.

6. References

Document	Source
[1] DVTDS Overview	www.teracortex.com/en/doc/DVTDS_DirectoryServer.pdf
[3] ELDC User guide	www.teracortex.com/en/doc/ELDC_UserGuide.pdf
[4] Extended LDIF	www.teracortex.com/en/doc/ExtendedLDIF.pdf Also available as Internet Draft at www.ietf.org
[5] Embedded LDIF	www.teracortex.com/en/doc/EmbeddedLDIF.pdf Also available as Internet Draft at www.ietf.org
[6] Embedded LDIF for C	www.teracortex.com/en/doc/EmbeddedLDIF_C.pdf Also available as Internet Draft at www.ietf.org
[7] LDAP Queue Length Control	www.teracortex.com/en/doc/QueueLengthControl.pdf Also available as Internet Draft at www.ietf.org
[8] Oracle OID benchmark March 2013	www.oracle.com/technetwork/middleware/id-mgmt/overview/oid-50m-user-sparct5-2-benchmark-1955392.pdf
[9] Aerospike benchmark at Intel labs	www.intel.com/content/www/us/en/processors/xeon/xeon-e5-v3-ssd-aerospike-nosql-brief.html
[10] Lynnlangit blog	lynnlangit.com/2015/01/28/lessons-learned-benchmarking-nosql-on-the-aws-cloud-aerospikedb-and-redis
[11] Foundation DB	foundationdb.com/key-value-store/performance
[12] DVTDS Replication Benchmark	www.teracortex.com/en/doc/DVTDS_Benchmark_GlobalReplication.pdf

7. Author

Christian Hollstein E-Mail: chollstein@teracortex.com

TeraCortex Phone: 0049 / 5473 / 9933

Hopfenbrede 2 Mobile: 0049 / 160 / 96220958

D-49179 Ostercappeln