

MySQL Cluster™ on Multi-Core Intel Xeon™ using Dolphin Express™ :

**DELIVERING 'REAL-TIME' RESPONSE TO THE
DATABASE MARKET**

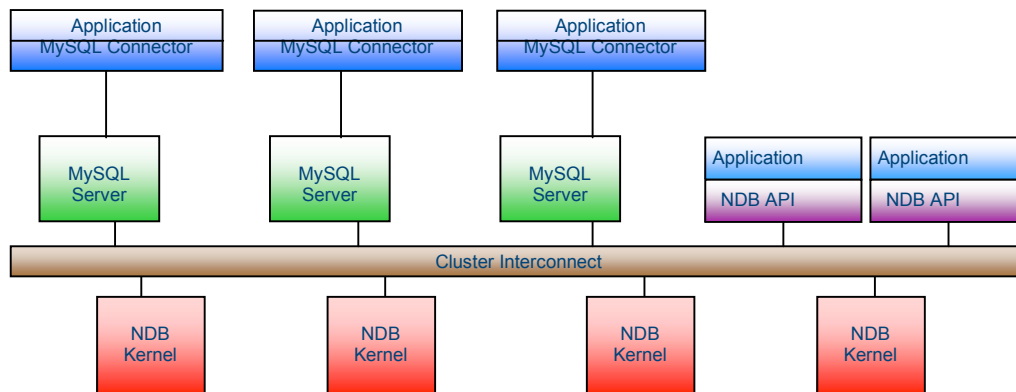
*Mikael Ronström, Marianne Ronström
Open Source Consultant
iClaustron AB*

1 Introduction

In this technical white paper we show how MySQL Cluster on Multi-Core Intel Xeon based on the Intel Core2™ micro-architecture in combination with the Dolphin Express cluster interconnect becomes a 'real-time' DBMS suitable for markets with extremely low latency or 'real-time' requirements. Low latency is important in delivering fast response time in any database application, but for some markets such as telecom, financial services and many embedded systems, this requirement is a critical requirement. Our results demonstrate that Dolphin Express provides a response time improvement of up to 292% as well as significant performance increases in high-load cases, particularly in larger clusters, making it ideal for any performance sensitive database application and uniquely able to meet the needs of 'real-time' database applications.

MySQL Cluster is an Open Source Database technology developed by MySQL AB that brings High Availability, Scalability and High Performance to database applications. In this technical white paper we show how it can also provide 'real-time' characteristics. MySQL Cluster uses the standard MySQL technology with a clustered database backend, NDB Cluster. MySQL Cluster can be accessed from the MySQL Connector's which provides access to the full LAMP stack; it can also be accessed from the NDB API that works directly with NDB Cluster.

MySQL Cluster



Given that MySQL Cluster is a clustered backend, communication performance is important and therefore the choice of cluster interconnect technology is very important. There are many aspects of the cluster interconnect that are important, but probably the most important is how well integrated the interconnect software is with MySQL Cluster and the latency in the interconnect hardware together with the hardware/software interface.

The *Dolphin Express* cluster interconnect architecture provides low latency on the hardware level in combination with highly optimized TCP/IP bypass software, called *SuperSockets*. SuperSockets are a fully compatible socket API implementation optimized for the underlying cluster interconnects hardware. Using Dolphin Express, any existing MySQL Cluster installation can be immediately and transparently accelerated. The socket implementation is well engineered:

- SuperSockets makes use of different schemes for performing write operations dependent on the data size to minimize latency for sending small data blocks while providing maximum bandwidth for large data blocks.
- For small messages up to 1kB, SuperSockets use an implementation of the CRC algorithm developed by Intel which is optimised for the Multi-Core Intel Xeon and which has a very significant effect on performance of the Dolphin Express solution.
- The tight integration of the hardware interrupt mechanism with the low-level software makes it possible to minimize the number of interrupts required for communication, saving valuable CPU cycles for the real work. The key to this is the possibility of the receiver to actively control interrupt delivery based on its current status. Basically, interrupts are only activated if a process blocks for a message. In contrast, Ethernet delivers interrupts with each incoming data block (MTU), or interrupts are statically throttled at the cost of higher latency.

These techniques have the following impact:

- In high load situations, the number of interrupts *decreases* with SuperSockets as the process rarely blocks for incoming data. The interrupt rate for Ethernet in this situation *increases* as at least one interrupt per data block is created, or the interrupt rate is throttled at the cost of higher latencies.
- For medium sized data blocks up to around 40kB, exactly one interrupt is created per delivery.
- Using the feature to use polling as described in chapter 7.1.4 will at high load situations completely remove all interrupt processing.

- Data blocks larger than 8kB are not copied within the memory of the sending node, but instead are directly copied to remote memory. This increases the transfer performance and avoids cache pollution, further increasing effective application performance.

These techniques are very important in making Dolphin Express a very efficient solution in high-load cases. This is particularly true in configurations where all CPU's are kept busy. Dolphin and MySQL have developed features in MySQL Cluster that leverages this efficiency for database applications. These features are referred in this paper to as the '*real-time*' extension of MySQL Cluster.

Furthermore for real-time markets it is not enough to simply be able to process many messages at peak loads. It is also important to have very low latency with very stable response times. As an example in financial markets many companies receive the same information simultaneously and the company that can process the information first has a first-mover advantage and has the best opportunity to make money for its clients. Faster response time of the DBMS is an important part of such information processing for the financial markets.

Given the significance of low response times, Dolphin and MySQL have developed a number of tweaks to the MySQL Cluster software that makes its '*real-time*' behaviour improve considerably. These are currently in the process of being integrated with the MySQL Cluster Carrier Grade Edition. Without using those tweaks, the Dolphin Express solution already decreases the standard deviation of response times considerably. With these optimizations implemented and properly used, the standard deviation further decreases significantly and the application behaviour becomes very predictable. Thus, the risk of losing economical advantage simply because of random effects in the software is decreased.

2 About Benchmarks

These benchmarks have been performed in a cooperative project between MySQL, Dolphin ICS and Intel. Most of the benchmarks were executed at Intel's Parallel Computing Center in DuPont, Washington. Some of the tests were executed using the same type of machines located in a MySQL lab.

The software developed for these benchmarks in the form of additions to the DBT2 benchmark developed by OSDL and a set of scripts to manage start and stop of various processes in MySQL Cluster is fed back to the open source community and can be downloaded from www.iclaustron.com. There is also documentation available on how to set-up everything to run these tests, when configured properly running a benchmark suite is one single command with a number of parameters.

3 Performance Aspects of Interconnect Hardware

The cluster interconnect architecture can improve a number of performance parameters. These are:

- 1) Latency Improvements
- 2) Efficiency Improvement in CPU-limited scenarios
- 3) Efficiency Improvement where there are abundant CPU resources
- 4) Bandwidth Improvements

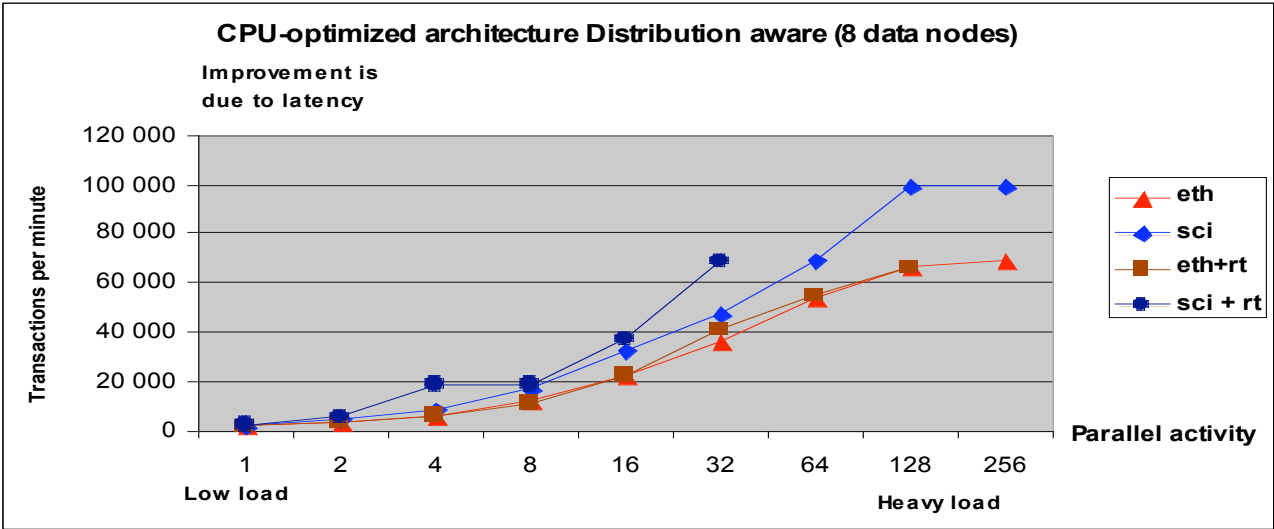
The reason we differ between efficiency improvements in CPU-limited scenarios and scenarios where there is abundant CPU resources are that they are affected by different factors.

3.1 Latency Improvements

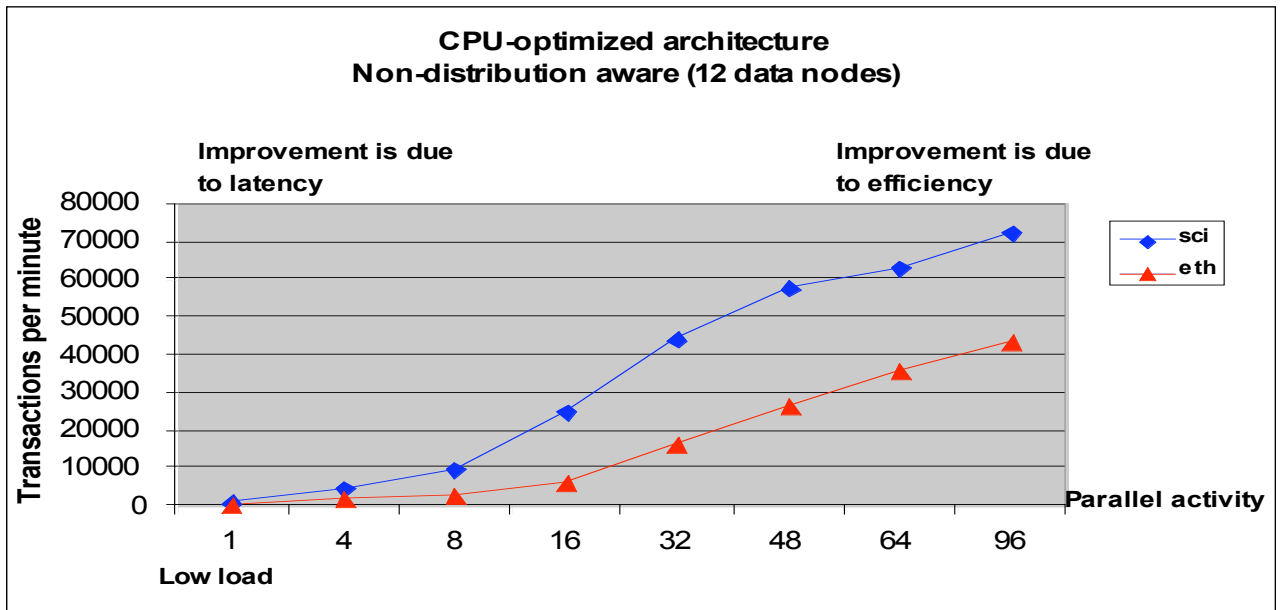
The improvement of latency has a great effect on response time and Dolphin Express achieves this through its integrated hardware/software solution. The improvement can be as great as a factor of four although a factor of two is more normal for database operations. In our benchmarks, we demonstrate this factor as very dominant when the number of parallel threads of activity is low.

3.2 Efficiency Improvement in CPU-limited scenarios

When the system is heavily loaded in a CPU-limited scenario the CPU resources required to handle communication are the decisive factor in the efficiency. As shown below Dolphin Express brings a factor of 40-70% efficiency improvement to the system. Thus Dolphin Express can support at least 40% more transactions in heavy load scenarios.

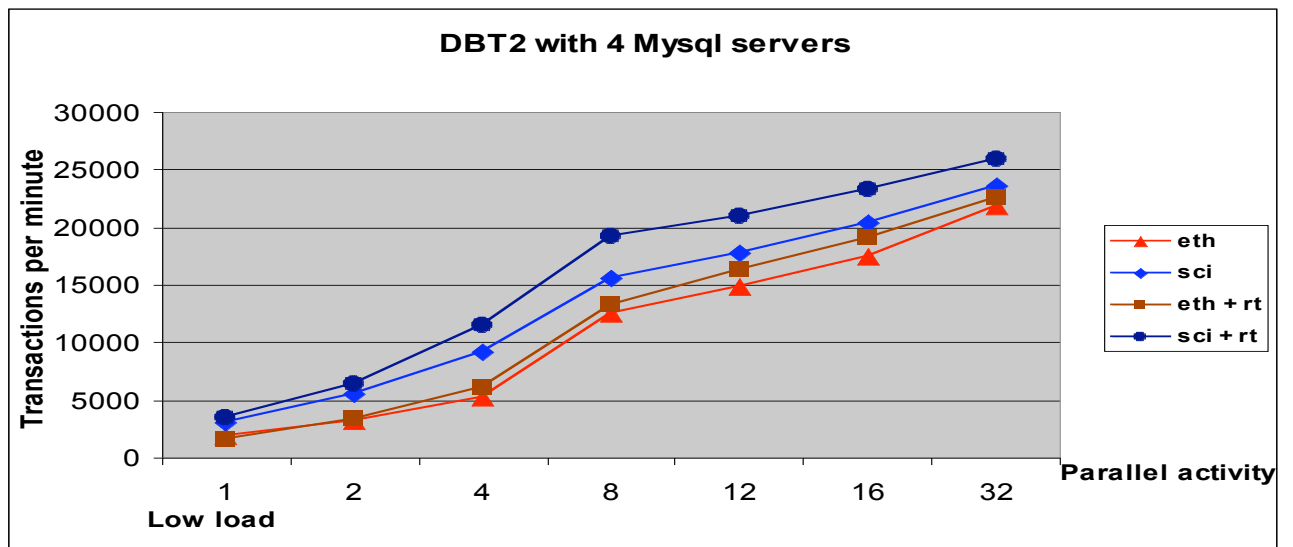


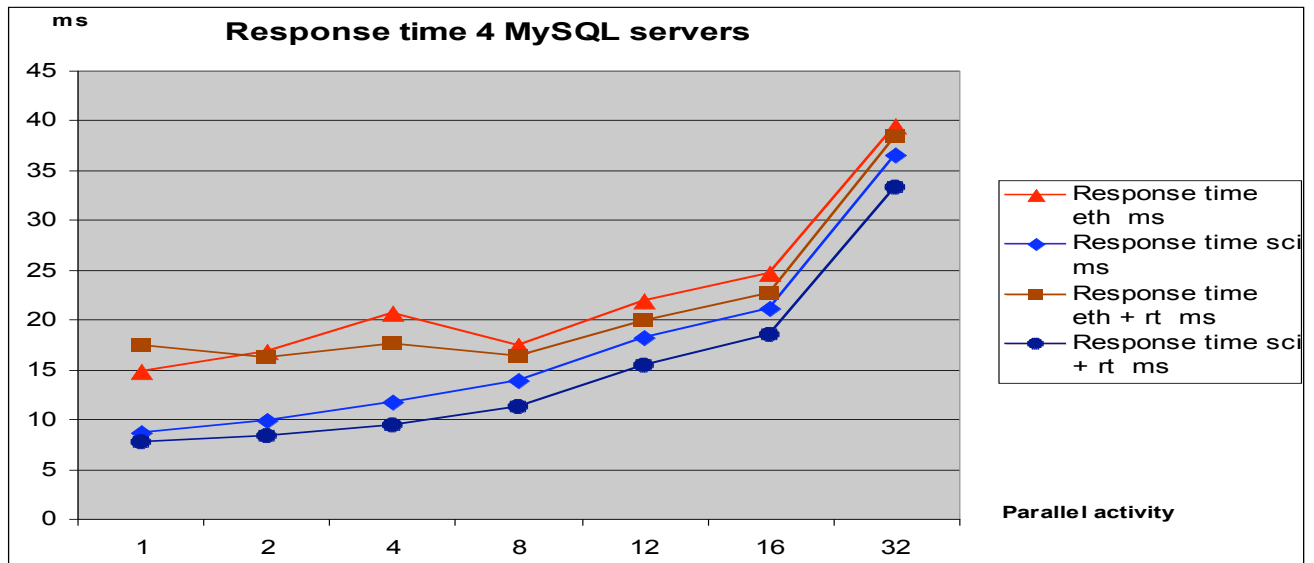
The graph above was in a scenario where the transaction coordinator is placed where data of the transaction is residing. In the graph below we show a similar case where transaction coordinator is randomly chosen. In the graph below we can see that the improvement by Dolphin Express can be as much as 292% in this case. The ability to choose where the transaction coordinator is located is a new feature currently being integrated in the MySQL Cluster Carrier Grade Edition.



3.3 Efficiency Improvement where CPU resources are abundant

When there are abundant CPU resources available, the decisive factor of the efficiency of a cluster interconnect depends on how it manages interrupts to the processor. In cases with high load and abundant CPU resources Dolphin Express still supersedes the Ethernet interconnect by at least 15% higher transaction rate, even in small clusters.





What we see in the above graphs is that up to 8 threads we can handle more transactions at a very good maintained response time. We improve throughput by 650% by only increasing response time by 50%. Whereas from 8 to 32 parallel activities we only increase throughput by 30% at the cost of a 200% increase in response time. Thus optimum for 'real-time' applications is here at 8 parallel activities where we can see that Dolphin Express using the 'real-time' extension have approximately a 50% performance advantage compared to Ethernet.

3.4 Bandwidth Improvements

The Dolphin Express hardware can handle 3-5 times higher bandwidth compared to Gigabit Ethernet. This is relevant in applications storing large objects in MySQL Cluster. In this technical white paper we have not made any benchmarks testing these improvements other than a simple bandwidth test on the cluster interconnects.

3.5 Conclusion

MySQL Cluster using Dolphin Express can be used to build a 'real-time' DBMS product with very high performance and very high availability. Our results demonstrate response improvements of up to 292% as well as significant performance increases in high-load cases, particularly in larger clusters.

4 Performance Aspects of Multi-Core Intel Xeon

The choice of the Multi-Core Intel Xeon improved performance by about 75% compared to a benchmark run with the same number of cores and a 2-node set-up in both cases. The new Intel Core2 micro-architecture has made possible very impressive results. Much higher performance was achieved at a lower frequency (2.67GHz vs. 2.8GHz) compared to previous Intel generation chip. Also use of power has been greatly improved. The new multi-core architecture is the enabler of many of the new performance features described later in this technical white paper.

5 Performance Aspects of MySQL Cluster

5.1 Distribution-aware application

The most important performance parameter in clusters consisting of more than 2 data nodes is whether the application is distribution aware or not. Actually applications in 5.0 could only be distribution aware if they used the NDB API. But a new feature has been developed that extends this also to the MySQL Server. As can be seen by comparing the benchmark result in 10.3.3 which is a distribution-aware benchmark and the benchmark in 10.3.4 which isn't distribution aware, the distribution aware benchmark delivers 40% more throughput with less data nodes. The difference is even bigger using Ethernet. No application will be able to handle distribution awareness perfectly so even distribution aware applications will have some parts where it needs data from other node groups than the common one.

For DBT2 by partitioning on warehouse we can ensure that almost all accesses within a transaction are within the same node group. The only exception is the accesses to the item table. The partitioning feature is new to MySQL version 5.1.

As seen here to write distribution aware applications requires both partitioning support and a new patch that will be integrated in MySQL Cluster Carrier Grade Edition in the next few months. Thus for current users of MySQL Cluster 5.0 the benefits of Dolphin Express is the benefit for non-distribution aware applications.

5.2 Configuration Parameters

It is important to set the proper configuration parameters. The most important one to set are:

```
--ndb-use-exact-count=0  
--ndb-index-stat-enable=0 (only 5.1)  
--ndb-force-send=1  
--engine-condition-pushdown=1
```

Interestingly here is that not setting these properly will mean more communication and thus more benefit from using Dolphin Express. This is a general principle that the use of Dolphin Express makes the application less susceptible to mistakes by application developers and DBA's. Thus use of Dolphin Express can decrease the application development time. More details on these parameters are provided later in this technical white paper.

6 Details of the Dolphin Express implementation

A very important reason why Dolphin Express brings such good performance to MySQL Cluster is that Dolphin has worked on the socket implementation for a couple of years and constantly improved it up to the point where it is now a mature solution with excellent performance for all socket applications. Dolphin Express has also been extensively tested and benchmarked using MySQL Cluster to remove any bottlenecks.

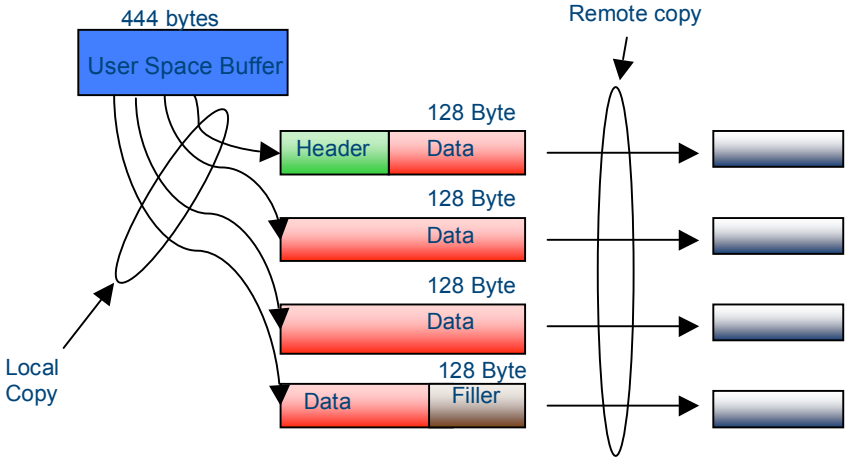
When doing writes to the sockets there are different algorithms that are optimal for various sizes of the message. The wake-up logic is implemented very efficiently to ensure that there are no unnecessary interrupts coming to the receiver side creating unwanted overhead (these interrupts generate more than 30% overhead on the NDB kernel in DBT2 tests). Interrupts that are needed to really wake-up the process on the receiver side are still issued. In contrast to Ethernet, interrupt processing isn't required to process any part of the network protocol.

Dolphin Express supports the use of multiple channels or multiple cards on the same box. The provided implementation gives only benefits (increased bandwidth for larger messages and redundancy for adapter, cable or node failures) without any negative side effects as they are known from channel bonding with Gigabit Ethernet. High Availability is integrated in the solution in such a way that one can get optimal use of multiple channels when they are all available and fast failover to only using the surviving channels when a channel fails for some reason. This is in contrast to teaming Ethernet drivers in Linux where one gets failover in 50 milliseconds except in cases where switches are involved when it can take up to 2 seconds, and the active-active approach gives about the same throughput as one Ethernet link when using Gigabit Ethernet.

6.1.1 Inline Protocol

The inline protocol has been developed to avoid high costs of checking for recoverable network connectivity errors (like cable pulled) as part of the socket write while ensuring reliable transport at the same time. The traditional integrity check takes about 4-5 microseconds which makes it very desirable for short messages to avoid this cost. The inline protocol solves this by adding a checksum to the message such that the receiver can validate that the message has been sent correctly and in its entirety. Checksums of messages are also used in hardware so the reason for using the checksum here is that the sender doesn't stay around to validate that the send operation has correctly and completely reached the destination buffer before signalling the new data to the receiver. Instead, a checksum is added to the self-synchronizing data write such that the receiver can validate the completeness and correctness of the data. If it detects an error, the receiver sends a message to the writer to resend the message. On the sender side, there is logic to ensure that messages are resent quickly in such error cases.

Given that errors on messages are extremely rare this protocol decreases the cost of small messages from around 6-8 microseconds down to about 3 microseconds. The overhead of calculating checksums does cause overhead on larger messages so this protocol is only used up to message sizes of 1008 bytes.

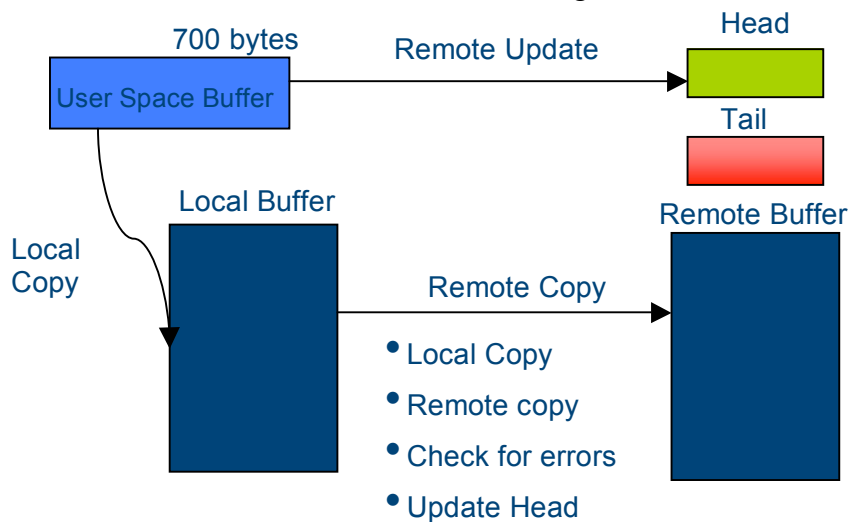


MySQL Cluster benefits greatly from this optimisation since there are many short messages in MySQL Cluster as part of the distributed transaction protocol. In the DBT2 tests the majority of the messages are using this protocol (98% of the messages).

The inline protocol will make efficient use of several channels if they are available by writing 256 bytes per channel at a time.

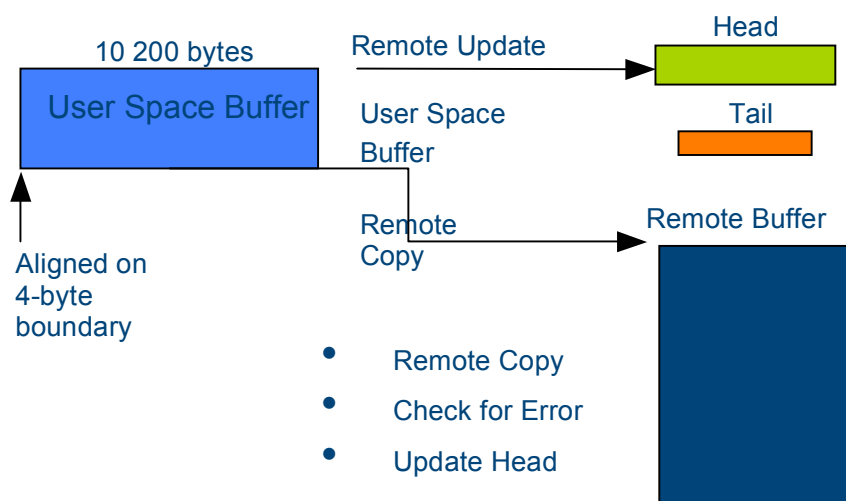
6.1.2 Short protocol

Longer messages, between 1009 and 8192 bytes, use the Short protocol, this protocol takes the message, copies it to a kernel buffer and prepares a message to send to the other side. Then the message is sent from the kernel buffer. After sending the message an error check is performed to ensure that the message has been safely delivered to the receiving side. The final step is to update the header information with a 4 byte write. This means that these short messages have a small overhead in copying the message but it avoids calculation of checksums and uses the hardware checksums to ensure that messages are properly delivered to the receiver. If several channels exist there is another limit where the extra channel is used to avoid the double error check for smaller messages.



6.1.3 Long protocol

The Long Protocol, for messages larger than 8192 bytes, is similar to the Short protocol except that the copy takes place directly from the user space buffer to the receiver side. This requires an extra write operation to transfer the header information and is thus avoided unless the message is fairly large. The Long Protocol can efficiently make good use of dual channels since the overhead of checking errors on both channels is won back by the benefit of writing to dual channels.



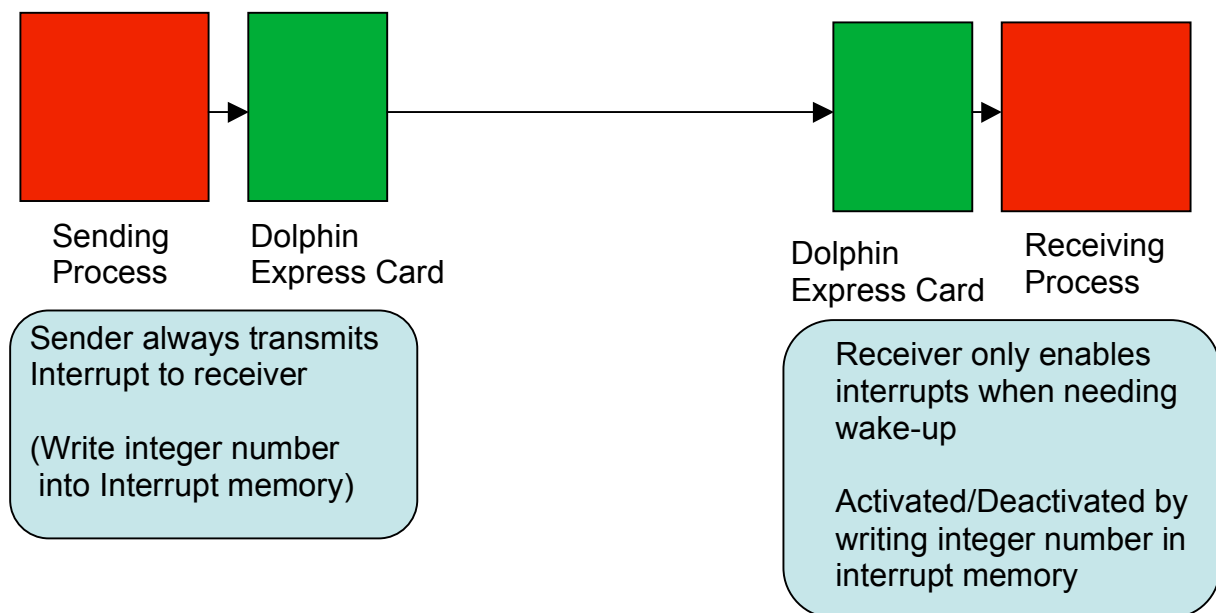
6.2 Handling of wake-up using Dolphin Express

The method to wake-up the receiver side is the same independent of the protocol chosen for sending the message. It always involves the writing of a 4 byte integer number to the receiver side. This write operation is received by the Dolphin Express adapter and generates an interrupt if the user is currently awaiting a message from this socket. The Dolphin Express driver will activate/deactivate interrupt reception based on what the application is doing. For interrupts to be triggered, the user must either be blocking within a read on the socket or by a select or poll call on a set of sockets, also use of epoll on the socket will enable triggers at all times. In all other cases the Dolphin Express adapter will ignore any incoming interrupt triggers and the receiver will get the data either when finally issuing a read on the socket or will discover the data when issuing a select or poll involving the socket.

Socket calls that enable interrupt reception are:

- select/poll calls with timeout > 0 where no data was available on the sockets
- Blocking read where no data is available on socket
- Use of epoll on the socket

Thus Dolphin Express will handle efficiently both the case of a 'real-time' configuration where the NDB kernel never sleeps and also a moderate configuration where the NDB kernel sleeps when no messages have been received.



6.3 High Availability solution for Dolphin Express

The various protocols will have slightly different manners of checking for error. With the Short and Long protocol it is very straightforward to simply resend the message using the non-faulty channel when an error is discovered by the error check routine. For the inline protocol the resend will be handled by some time-outs on a small number of milliseconds. Thus failover of network traffic will always happen within a few milliseconds in failure cases.

7 Performance Tuning of MySQL Cluster

MySQL Cluster is a clustered storage engine integrated within MySQL. Any application using the LAMP stack can also be ported to use MySQL Cluster by simply changing the

storage engine of the tables in MySQL. Naturally not all applications will benefit from this change but the usage of Dolphin Express will increase the number of applications which will benefit from this.

The storage engine in MySQL Cluster is called NDB Cluster. This implements a parallel storage manager with very high performance and very high availability. It has been very successful within telecom applications. In those applications the performance and availability features enables it to become THE Telecom DBMS.

In this technical white paper we show how some small changes to MySQL Cluster in combination with use of Dolphin Express will make it very useful also for applications where short response times are critical. An example of such an application is the financial market where market players receive the same information simultaneously and those that will process the information in the quickest manner will have an advantage in making choices about what to buy and sell. We also give lots of other advices on performance tuning of MySQL Cluster such as configuration parameters and hardware selection.

If anyone is interested in trying out those new features even before they've been integrated into the MySQL Cluster Carrier Grade Edition a source download of a special benchmark version is available at www.iclaustron.com. Together with this download there is also a set of other downloads available to be able to run the DBT2 benchmark in an easy manner, including documentation of how to set-up everything.

7.1 Upcoming features in MySQL Cluster Carrier Grade Edition

7.1.1 Setting threads to Real-time priority in NDB Cluster

Traditional process scheduling in a time-sharing operating system like e.g. Linux, focus on sharing CPU resources between processes in the system. This has the effect that process priority decrease after executing for a while. It also means that response times are very variable over time. For MySQL Cluster this means that as load goes up the predictability of response times goes down since the process will receive lower priority after executing at high load for a while. The way to overcome this is to use real-time scheduling algorithms.

In Linux real-time scheduling algorithms can be applied on a thread level. This is not portable to all different operating systems but is a very useful feature. In the NDB kernel there is one thread, the main thread that does the entire database related execution. This thread implements its own scheduling algorithm. The other threads implements file system operations, setting up socket connections and a watch-dog thread. It is important that these threads have higher priority than the main thread; otherwise the main thread would potentially block those threads from ever executing.

A new configuration parameter has been introduced in MySQL Cluster that can set this feature on or off. This parameter can also be changed dynamically through the management client. This is important if real-time behaviour is only needed during certain time-periods of the day. It might be that during a backup or other similar operations the database administrator wants to provide better chance for backup software to make use of the CPU's for a well-defined period of time.

Remember here that the purpose of real-time scheduling is to ensure that the NDB kernel has a very predictable response time, not only to increase throughput. This predictability will also

have the positive side effect that availability increases since the risk of problems at high peak loads diminishes.

A warning here is to not use this feature without proper testing since real-time has a higher priority than the operating system activity and particularly in CPU limited configurations can cause issues. Thus it's only recommended for use in cases where each data node can have two cores available.

7.1.2 Locking NDB Kernel to Main Memory

To avoid page misses is a critical feature for any 'real-time' application. To avoid this, some operating systems implement the possibility to lock process memory in main memory ensuring that the process memory is not swapped out on disk. This feature already exists in MySQL Cluster.

7.1.3 Locking thread(s) to a CPU in NDB Cluster

At high loads it is more likely that the operating system scheduler decides to move threads from one CPU to another. Whenever this happens it increases response times, decreases performance and makes response time less predictable. To overcome this one can lock threads to CPU's in some operating systems. In Solaris one can even ensure that the thread(s) get exclusive access to CPU's. Using this feature can in some configurations increase 'real-time' characteristics of MySQL Cluster considerably.

This ensures that the execution always makes use of the same CPU and its caches, thus decreasing the variability due to switching the threads between the CPU's in the box. It can also increase the variability if there are many other processes competing for the same CPU as the thread is executing on. So this option is mainly interesting in the context of a system where the NDB kernel has exclusive access to a part of the resources of the computer.

To handle this two new configuration parameters have been introduced in MySQL Cluster. The first one sets the CPU id where to lock the main thread and the second parameter specifies where the remaining threads are to be locked to. This means that one can ensure that the NDB kernel executes only on those two CPU ids and nowhere else in the computer. This makes it safer to mix more than one NDB kernel in the same computer or mixing an NDB kernel and a MySQL Server in the same computer.

What this feature mainly achieves is that it tells the operating system that although the thread locked to a certain CPU might on a short-term basis be decided for switching back and forth between CPU's, this should not occur since the thread is more important than other threads even though the others might look more appealing on a short-term basis from time to time. Thus this feature also removes variability from the performance figures.

An important note here is to ensure that the main thread isn't locked to the CPU where most interrupt activity is. In many cases CPU 0 is used for lots of interrupt processing. To see where interrupts are processed type the file `/proc/interrupts` in Linux. Linux also has some ability to control where interrupts are placed for a specific IRQ line. This is performed by setting the bitmap in `/proc/IRQ/169/smp_affinity` (here 169 is an example of the IRQ line, `/proc/interrupts` specifies the IRQ for each driver). Not all drivers will respect the settings in the `smp_affinity` file.

7.1.4 Keeping main thread awake in NDB Cluster

There is also another change that has been made to the NDB kernel. This applies to how the execution of the main thread is handled. Currently it runs an algorithm as shown below:

- 1) Check if any data from Sockets
- 2) If data available Read from Sockets
- 3) Check if any timed messages are due for execution
- 4) Execute Jobs
- 5) Send Data gathered in Send Buffers on Sockets
- 6) Go back to 1)

The sleeping in the main thread happens in step 1) above when there are no jobs left after doing step 4). The main idea with the change to increase real-time predictability is to avoid the sleep in step 1). This is done by introducing a new parameter that specifies the minimum time that one will continue looping while not finding any data on sockets, before going to sleep.

The reason why this increases predictability is that it avoids context switching and interrupts. Context switching and interrupts means a significant extra overhead in CPU resources which is naturally good to handle multi-workload systems. Here we do however aim for maximum predictability by turning the computer into a single workload system and then context switching should be avoided unless absolutely needed and in particular the main thread of the NDB kernel should avoid context switching to ensure that its data and instruction caches stay focused on executing the NDB kernel code.

Another benefit of not sleeping is that the time to start servicing a new request is decreased since no time is spent scheduling an interrupt, executing the interrupt and waking up the thread for execution. All of these delays can be removed, providing better `real-time` characteristics and better system throughput.

7.1.5 Extra Job buffer execution before sending

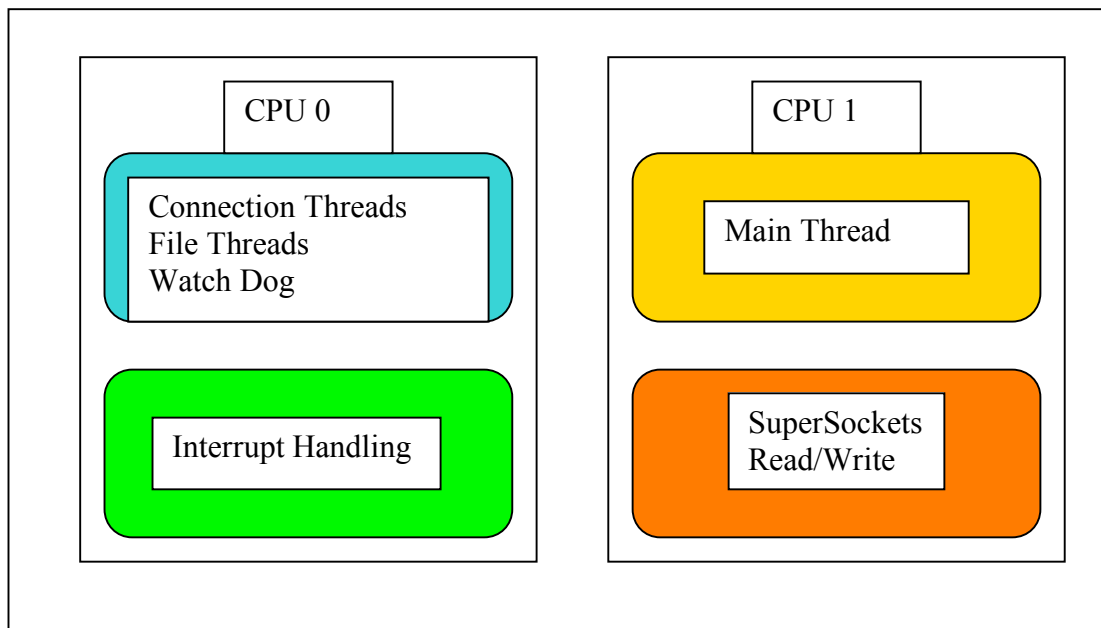
In the loop provided above another possible optimisation is that after step 4) to receive new data and if data is available execute it before sending any data. This has the effect of increasing message size over the cluster interconnect to improve efficiency. It has the disadvantage of increasing the delay in sending so should rarely be used in `real-time` applications.

7.1.6 Getting the best `real-time` behaviour

The very best `real-time` behaviour is achieved by the following use of the new parameters.

- 1) Use Dolphin Express for communication
- 2) Set parameter to Lock Main Memory to avoid Swapping (existing configuration parameter)
- 3) Set parameter to lock main thread to CPU id 1
- 4) Set parameter to lock maintenance threads to CPU id 0
- 5) Set parameter that Main Thread not go to sleep until 200 microseconds have passed (means main thread will never sleep unless activity goes to a very low rate)

The above scenario (displayed in figure below) assumes a dual-core computer and similar reasoning can be applied on computers with more cores. Using e.g. a quad-core box it is a good idea to place one data node per core except on CPU 0 where one sets the interrupts to occur on this CPU. Using Dolphin Express the interrupts can be avoided and thus in this scenario one can place four data nodes on the computer, one per core.



Setting real-time priority won't affect the throughput in any significant manner; it should be used if safe to do so. These parameter settings provide the NDB kernel with a proper real-time environment and will ensure that responses from the NDB kernel will be very predictable. These parameter settings can only be used on a machine that has at least two cores and the machine should be completely dedicated to running the NDB kernel, no other programs should be running on the machine. This document doesn't go into details of how to create such a setting. Further improvements can be achieved by properly configuring the OS installation. If the machine has more than two cores also other programs can be executed on the machine or possibly several data nodes. In a 'real-time' environment it is however recommended to not mix different applications on the same box. It is possible however to dimension the system for several data nodes and the features described above makes it possible to divide the cores amongst the data nodes.

If these settings are used on a quad-core box one could potentially use another MySQL Server or another NDB kernel on the same box. In this case it is important to ensure that they don't use the same CPU resources. This can be achieved by locking the MySQL Server onto CPU id 2 and 3 to ensure that it doesn't coincide with the NDB kernel on CPU id 0 and 1. Using Dolphin Express will ensure that communication is extremely timely and as has been shown in a previous section these settings means that the NDB kernel node will not receive any interrupts at all and thus will not spend any time on unnecessary context switching.

These settings ensure that performance will be very stable even in high load scenarios. The settings mean that no special problems occur at high loads other than a higher load on networks and disks. Thus it is important to dimension the hard disks such that they can handle the highest loads that the node can handle. The system should always be dimensioned such that the CPU's are the bottleneck in the system.

7.1.7 Removal of Reads before updates in MySQL Server

Whenever the MySQL Server performs an update or a delete it does first perform a read of the data. This is to enable all the necessary checks of WHERE-clause, to enable calculation of

expressions in the SET part of the UPDATE statement. There is also a set of other special cases where it is necessary dependent on the storage engine used. In some cases it is possible to remove this read before the update. This is the case if the access is through a primary key or unique key and the SET statements only assign constant values. There are a lot of conditions on where this optimisation cannot be used but there are a sufficient number of cases where it can be used to make it interesting. For the DBT2 benchmark avoiding those reads increases performance by about 10%. In DBT2 actually all UPDATE's and DELETE's fall into the category that can be optimised.

Reasons of when this feature is not used in MySQL Cluster is when it needs to read some part of the record that isn't part of the key provided in the query. Another reason is when updating of a unique index occurs in the query and also deletes in a table that have BLOB's of any kind. Finally also all queries involving BLOB's cannot use this optimisation.

7.1.8 Enabling writing Distribution Aware Applications

To enable better performance in clusters when there are more than 2 data nodes it is important to make the application distribution aware. Using 4 data nodes it is possible to increase performance by 30% in this way. In clusters where the number of data nodes is 8 and upwards the benefit is even greater, as high as 100-200% in some cases. Thus it is very important to understand how to write your application distribution aware.

In the current implementation of MySQL Cluster the selection of transaction coordinator for a transaction uses a round-robin algorithm except when there is a data node on the same machine in which case the transaction coordinator is placed on this data node.

For many applications, the DBT2 benchmark an excellent example, most transactions are executed using the same type of data. In the case of DBT2 all transactions are involving just one warehouse. The only exception is the access of an item table which is common for all warehouses. Most applications are not that perfectly partitioned, but in most applications there is a lot of locality in the transactions. By placing the transaction coordinator in proximity of the data used in the transaction one gets a distribution aware application. Using the NDB API this has been possible since version 2.1, now it has also been implemented using the MySQL Server. The selection of the transaction coordinator will be based on the first record access. Thus if the first access reads using the primary key, the transaction coordinator will be placed in the same data node where the primary replica of this record resides. This means that it is important to ensure that the first record access in the transaction is using the data where one wants the transaction coordinator placed. The MySQL Server will discover the node to use by checking which partition the access will use. This partition is then mapped through internal data structures to the node where the primary replica resides.

EXPLAIN PARTITIONS on the first query in a transaction and ensuring that it uses the proper partition and only one partition is an important tool in ensuring that you're application becomes distribution aware.

To write a distribution aware application it is necessary to use the new partitioning feature in MySQL 5.1. Using the DBT2 as an example, the DBT2 tables have composite keys where warehouse id is one of the keys. Thus using default tables in MySQL Cluster data for one warehouse would be spread all around the cluster. By partitioning on warehouse id in all tables where it is possible we get the desired effect. For this one can essentially use any of the partitioning variants although very few reasons can be found in this case to use any other than

PARTITION BY KEY or PARTITION BY HASH. The KEY partitioning is the default partitioning used in MySQL Cluster and the most integrated partitioning scheme. It uses a hash function local to MySQL Cluster. In this case for DBT2 the best partitioning scheme is to use PARTITION BY HASH (warehouse_id), the reason is that data is perfectly spread among the partitions whereas the data partitioning using KEY partitioning can be slightly unbalanced given that the number of warehouses are fairly small (around 5-10 per data node in most benchmarks we've executed).

To use the HASH partitioning scheme one needs to start the MySQL Server with the configuration parameter --new that enables a set of new features in the MySQL Server.

7.1.9 Using epoll in Linux for higher scalability

The implementation of MySQL Cluster uses the select system call to discover if any data is available on a set of sockets. The select has a fairly fixed overhead of a few microseconds plus a variable overhead of about half a microsecond per socket. Thus when the number of sockets grows to around 40-50 the time to do a select call has more than doubled compared to smaller clusters. This has a dampening effect on the scalability of applications in MySQL Cluster. In Linux there is a call that avoids this dynamic overhead. It's called epoll and consists of a set of three systems call, epoll_create, epoll_ctl and epoll_wait.

In this feature it is possible to mix sockets using poll and epoll. By setting NoPollSocketsInEpoll to 0 one uses only epoll, by setting it to 255 one uses poll for all sockets (poll has same dynamic overhead as select). By setting it to 2 the two most active sockets are using poll and the rest is using epoll. For events to be discovered by epoll, an interrupt is required. Thus for device drivers that can avoid interrupts as Dolphin Express it is possible to avoid interrupts on the most active sockets in this manner.

Using this feature it is possible to scale MySQL Cluster to hundreds and even later thousands of nodes in the cluster.

7.1.10 Enabling scaling of number of data nodes to 128

With the development towards higher number of cores in CPU's it is important to be able to have many more data nodes in MySQL Cluster. Thus the limit for number of data nodes have been raised to 128 and the limit on the number of nodes in the cluster are 255. On the roadmap for future MySQL Cluster releases is multithreaded data nodes but this feature enables clusters to grow to around 400 cores which should be sufficient for a while.

7.2 Important Configuration Parameters in MySQL Server

There are quite a few configuration parameters for MySQL Cluster that affects the operation of the MySQL Server. Here we only list the most important ones from a performance point of view and their use.

7.2.1 ndb-use-exact-count

This parameter if set turns SELECT COUNT(*) FROM TABLE; into very fast operation essentially just reading the number of records in table from variables. If not set this operation calculates the number of rows by selecting all records and counting them. The side effect of setting this variable is that the counting is performed for each query. This means that queries like SELECT * FROM TABLE WHERE pk=1; have an overhead of around 100%. Thus in applications like DBT2 it is absolutely advised to ensure this variable is set to 0 except

possibly temporarily setting it if the application issues a `SELECT COUNT(*) FROM TABLE;-query`.

7.2.2 engine-condition-pushdown

This means that processing of the WHERE-clause can be pushed down the data nodes in MySQL Cluster. For applications where the WHERE-clause isn't so complex this has a positive effect and for most queries scanning without indexes it can have a substantial effect. For DBT2 it has led to a speed-up of about 10%.

7.2.3 query-cache-size

In applications like DBT2 which updates tables all the time, the query cache is of no use. Thus disable it in those cases.

7.2.4 log-bin

In MySQL Cluster replication happens in special MySQL Servers that should only handle replication. Thus the normal MySQL Servers used for queries should not have the binlog enabled. If it is necessary to enable for other storage engines then ensure that the database(s) used by MySQL Cluster is skipped in the binlog (see MySQL manual on how to set this up).

7.3 Important Configuration Parameters in NDB configuration

The configuration parameters that are possible to set in MySQL Cluster is a long list. Here we will only focus on the most important ones for performance reasons.

7.3.1 DataMemory

This is the space required for the records in memory-based tables in MySQL Cluster. It is also used for non-unique indexes and for the unique index tables it is used for the record part. In the DBT2 tests, each warehouse consumes around 100 MByte of Data Memory. DBT2 also inserts records as part of the benchmark runs and thus it is necessary to have some space available for expansion. To handle a benchmark with 16 warehouses it is advisable to only fill DataMemory up to about 60% of the DataMemory. Thus total DataMemory in cluster should be 2700 MByte. Thus if NoOfReplicas is set to 2 this means that Data Memory per node should be $2700 \text{ MBytes} * 2 / \#Nodes$. In a cluster with 4 data nodes it is required to set DataMemory to 1350 MBytes. Version 5.0 of MySQL Cluster requires more memory; the above numbers comes from using version 5.1.

7.3.2 IndexMemory

This is the space required for the hash indexes in MySQL Cluster. Each record requires about 17 bytes of storage space. Small tables have an additional overhead since memory is allocated in pages. This memory is also used for unique indexes; each unique index is actually a normal table with the unique index as primary key and the primary key as attributes of the table. For DBT2 one requires 10% of the size required for the DataMemory. So in the example above one requires 135MByte.

7.3.3 NoOfFragmentLogFiles, DiskCheckpointSpeed

Number of REDO log files is an important parameter since if it is too small; updates will no longer be accepted until checkpoints have been able to move the tail of the log forward. A rule of thumb is to calculate enough REDO log space for 6 checkpoints. In the worst case this means that the entire DataMemory is to be sent off to disk as part of a checkpoint. The speed of these checkpoints is set by the parameter DiskCheckpointSpeed. So as an example if the

DiskCheckpointSpeed is set to 4M it means that 4 MByte per second will be checkpointed. Thus if DataMemory is 1350 MByte in size, worst case a checkpoint will take 338 seconds. Thus six checkpoints will take about 33 minutes. Thus the REDO log should be long enough that it will roll around after 33 minutes of activity at peak throughput of the system. By increasing DiskCheckpointSpeed we decrease the size of the Redo Log but also increase the base load on the CPU and the disks to handle checkpointing. Increasing the DiskCheckpointSpeed will also speed up cluster restart.

In most DBT2 configurations a number of 100 (100 * 64 MByte of Redo Log Space = 6.4 GByte) have been sufficient. If disk space is abundant there is no reason to not set to 300 which is a bit more conservative. Also if one has really large memory and many warehouses one should either set it higher or increase the DiskCheckpointSpeed.

7.3.4 RedoBuffer

Before redo logs are sent to disk they are stored in the RedoBuffer. This needs to be large enough to sustain all outstanding log writes not yet written to disk. The RedoBuffer can be reused when the OS reports back a successful write. The default value of 32 MBytes has proven to be too small for DBT2 tests in some cases. Thus a more conservative figure of 64 MBytes has generally been used.

7.3.5 DiskSyncSize

This configures how often the disk writes are made with a synch flag. It is important to not write without synch flag for too long in Linux since when the buffer memory is fully used the disk writes all have to wait for disk writes of previous file writes and the performance in such situations can easily bring down the speed of the database to half of the normal speed. Generally the default value should be ok but both larger and smaller values have been used.

7.3.6 LockPagesInMemory

It is important to lock the database process into memory to avoid swapping which can easily bring down the system. This feature requires that the ndbd process is started as the root user.

7.3.7 SchedulerSpinTimer

This is described in section 7.1.4. This variable can be set dynamically through the command “all dump 506 200” in the management client, 200 here is the number of microseconds that the scheduler will spin before going to sleep. This setting will not survive a node restart as is the case for all similar examples below.

7.3.8 SchedulerExecutionTimer

This is described in section 7.1.5. This variable can be set dynamically through the command “all dump 502 50” in the management client, where 50 is the number of microseconds the scheduler will execute before sending.

7.3.9 RealtimeScheduler

This is described in section 7.1.1. This variable can also be set in all data nodes by the command “all dump 503 1” in the management client. This can only be used if the ndbd process has been started as root user.

7.3.10 LockMaintThreadsToCPU, LockExecuteThreadToCPU

This is described in section 7.1.3. To lock the maintenance threads to CPU 2 one can also use the command “all dump 505 2” in the management client. To lock the main thread to CPU 3 in all nodes one can issue the command “all dump 504 3” in the management client. These settings can only be used if the ndbd process has been started as root user.

7.3.11 NoPollSocketsInEpoll

This is described in section 7.1.8. This variable can also be set in all data nodes by the command “all dump 507 255” in the management client, where 255 is the number of sockets to use poll instead of epoll.

8 Details of improvements due to Multi-Core Intel Xeon

There are many features of the Intel Core2 micro-architecture that enables many of the outstanding improvements that have been made possible through the use of new features in the MySQL Cluster Carrier Grade Edition and in the inside of the Dolphin Express implementation.

8.1 Intel Wide Dynamic Execution

The data nodes in MySQL Cluster are designed with a message passing model where software is divided into modules that communicate using messages. These modules send messages internally and externally. This provides an extremely efficient implementation removing all operating system locks (mutexes). In addition memory is statically allocated to avoid repeated memory allocation and deallocation that provides for a ‘real-time’ experience when using MySQL Cluster. However this software architecture leads to quite a few pack/unpack of signals. In C++ copying data between variables must be implemented as load/store pairs unless the program declares special variables to assist the code generator.

The Intel Core2 micro-architecture handles this elegantly by the enhanced Intel Wide Dynamic Execution technology - a combination of out-of-order execution, superscalar execution, data flow analysis, speculative execution and macro-op fusion.

Thus even when the compiler generates a set of load/store pairs the actual execution will perform all the loads at first followed by a set of store instructions thus providing very substantial performance improvements to the data nodes in MySQL Cluster.

8.2 Multiple Cores coupled with Intel Advanced SmartCache

The Multi Core Intel Xeon series processors deliver fully parallel execution of multiple software threads using two or more “execution cores” contained in a single Intel processor package. In addition, Intel's Advanced Smart Cache is optimized for multiple cores allowing efficient access to frequently used data.

Multi-core processors enable quite a few new possibilities for MySQL Cluster. Examples of these are locking threads to CPU’s and shielding them from other activity. Some of these new features are described in this paper. The architecture of MySQL Cluster is very well prepared to take advantage of additional cores with extended software features as the number of cores increase. Many of these features are already at the drawing board.

In addition, the larger and more efficient cache memories had a very significant effect on scalability in number of threads of the MySQL Server. In test runs using previous generation

Intel servers the MySQL Server scaled to 6-8 threads in the DBT2 tests. In the tests executed on the new Multi Core Intel Xeon processors the MySQL Server scaled to 12-16 threads.

We look forward to showing even better performance on future versions of the Multi Core Intel Xeon series platforms as the number of cores increase, new performance levels are enabled by the platform, and the software is optimized to take advantage of the new architectural features.

8.3 Improved CRC algorithm

A new CRC algorithm developed at Intel's Corporate Technology Group, called "Slicing by 8" improves CRC performance dramatically. The new CRC algorithm proved very useful in improving efficiency of short messages in the Dolphin Express driver. The implementation is optimized for the Multi-Core Intel Xeon architecture and in our tests provided about twice the performance on the CRC calculation (a major part of CPU cost for short messages).

The source code for the "Slicing by 8" CRC algorithm is available on sourceforge.net under a BSD license, contributed by Intel.

9 Hardware Selection for MySQL Cluster

When tuning performance the baseline for what performance one can get is achieved already in the selection of hardware. As we have already shown buying cluster interconnects will have a very good payback, especially in larger clusters. However choice of computers, choice of memory sizes, choice of disks and the number of them is also essential parameters.

9.1 System Choices

9.1.1 Minimal system

The very minimal system one wants contains 3 computers. Two of those needs to be proper server machines and the last one simply need to be a computer that can answer one question per crash. These two machines obviously need to run both data node(s) and MySQL Server(s). For an application that requires no performance one can probably survive with 2 single-socket dual core computers but for any performance hungry application quad cores are recommended. It is a very good idea to partition the cores, the memory and the disks to ensure stability of the system.

9.1.2 Maximum 2-node system

A more suitable architecture for MySQL Cluster is to separate the data nodes from the MySQL Servers. The reason is that they really need very different configurations of the computer. For a cluster with 2 data nodes it is enough to have 2 systems with a single-socket dual core CPU. However these data nodes need to be crammed with lots of memory and enough hard disks to sustain the load and keep all data in memory.

The machines running the MySQL Servers are quite differently. A good idea is to use about 2 cores per MySQL Server and separate those from each other so that they don't collide in cache memory usage. These machines have very limited requirement on memory, most MySQL Servers for MySQL Cluster fit very well within 250-500 MByte of memory space so even 2 GByte of memory for these computers would be sufficient. However they require fast CPU's. 2 quad core computers to balance the 2 data nodes is in many cases a proper balance.

9.1.3 Larger cluster

When building larger clusters it is usually a good idea to first scale-out the computers such that one uses the optimum server size. This is currently using dual-socket servers with the Multi-Core Intel Xeon. Such a server is today equipped with 2 dual-core or 2 quad-core CPU's. For data nodes it is sufficient to use dual-core CPU's since each core can handle one data node and generate very high load on memory, disks and interconnects. For the MySQL Servers it is better to use quad-core CPU's. Using Dolphin Express one can then have 4 data nodes per computer whereas with Ethernet it is only recommended to use 3 data nodes per computer given that one CPU should be allocated for handling Ethernet interrupts.

Thus a proper base block in building scalable large clusters is to use quad-core computers equipped with lots of memory and disks for the data nodes and as many computers for the MySQL Servers but using 8-core servers instead.

Some of the benchmarks reflect this configuration very well. We have used only quad-core boxes here. Thus for the 8-node cluster we used 2 computers for data nodes and 4 to handle the MySQL Servers. The 12-node system used 3 computers for data nodes and 6 computers for the MySQL Server nodes. Thus using 8-core computers for MySQL Server would have provided a proper mix with as many data node computers as computers used for MySQL Servers.

It would work fairly well to build clusters using computers with fewer cores instead but more computers mean more management and thus more work. Larger computers would get off the path of servers at optimum price and thus price per computer would be too high.

With the patch to handle up to 128 data nodes it should be possible to handle clusters with 32 computers for data nodes and 32 computers for MySQL Servers. Each data node in a server can easily handle 32 GByte of memory to a good price and thus we can build clusters with 1 TByte of memory in the data nodes from off-the-shelf components.

9.2 Choice of CPU

Most of the benchmarks have been executed on the Dual-Core Intel Xeon 5100. The servers have been based on Intel computers with dual sockets.

Our experiment verified that the new Xeon's outperformed the previous Intel generation of CPU's by as much 75% even though the frequency on the chips actually was lower than the previous generation. This makes it obvious that this new generation of Intel processors is an excellent choice in building a high performance MySQL Cluster.

9.3 Choice of Memory

As already mentioned very little memory is required on the MySQL Servers. On the data nodes it is important to cram as much memory as is required by the application. The data nodes can either be limited by memory or by CPU performance. If the application is memory bound it is better to build the system with servers with only one dual-core CPU and up to as many as 128 of those. With 128 machines it is affordable to build a cluster with 4 TByte of memory. A balanced choice could be to use the same type of computer with one socket using a dual-core CPU but running 2 data nodes on this computer. This provides more efficient use of CPU resources but less amount of memory per data node.

9.4 Choice of Disks

In most cases it is a good idea to have at least 1 disk per core to be able to sustain the load generated by writing the REDO log and by the checkpointing. Also the REDO log and the checkpoints need some disk space as well as backup disk space is required. For most systems one disk per data node should be sufficient but it is possible when using the NDB API to generate such high throughput that not even 1 disk can sustain the load generated from one data node. One data node can handle up to 100.000 updates per second of around 100 bytes per record and thus generating 30 MByte per second of REDO log writes in addition to the checkpoint writes. Most hard disks cannot sustain such a load when generated from the NDB kernel.

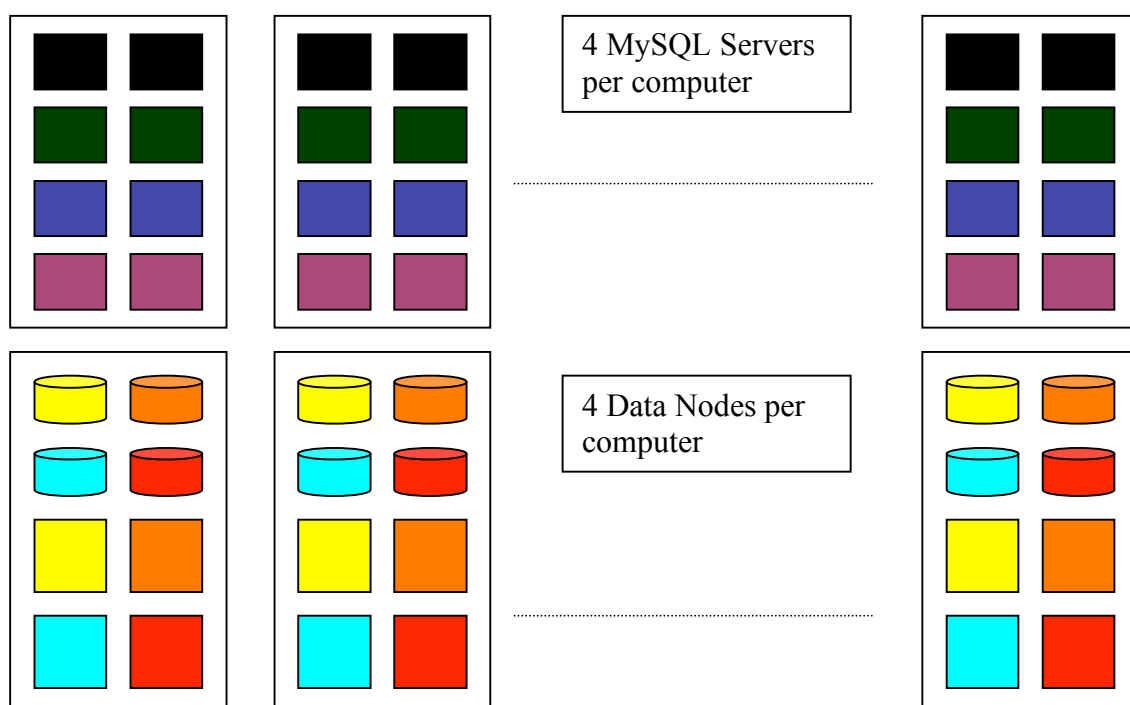
This is only applicable for a very efficient application using the NDB API, for the majority of applications one disk per data node should be sufficient.

9.5 Choice of Cluster Interconnect

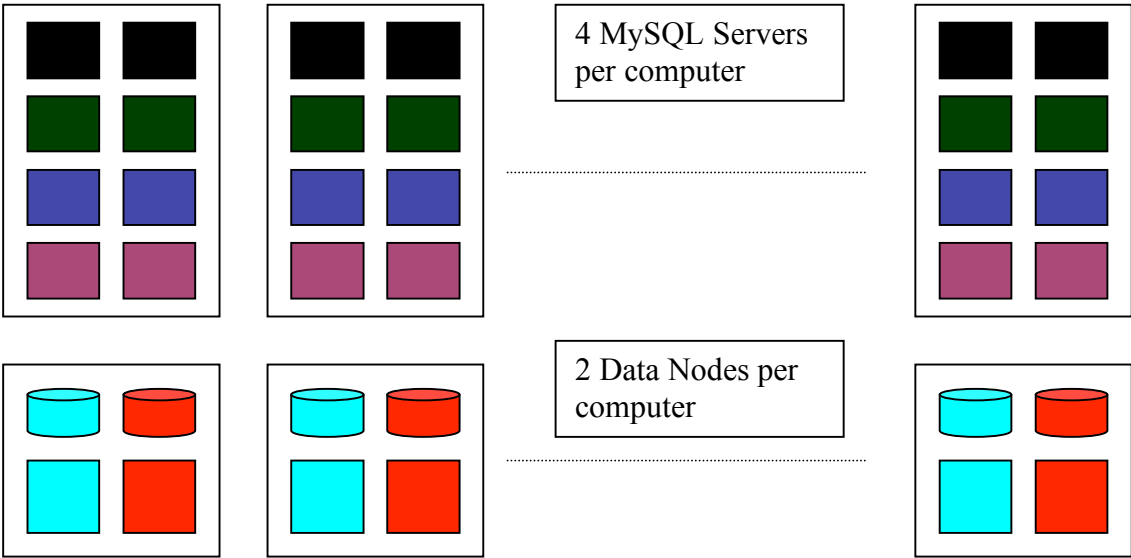
The impact of the choice of cluster interconnect has been very well covered in this technical white paper and given the discussion about the preferred servers used for building clusters it is clear that there are good economics in buying Dolphin Express cards for the servers since they provide so much better response time and also higher throughput, especially so in larger clusters.

9.6 Conclusion

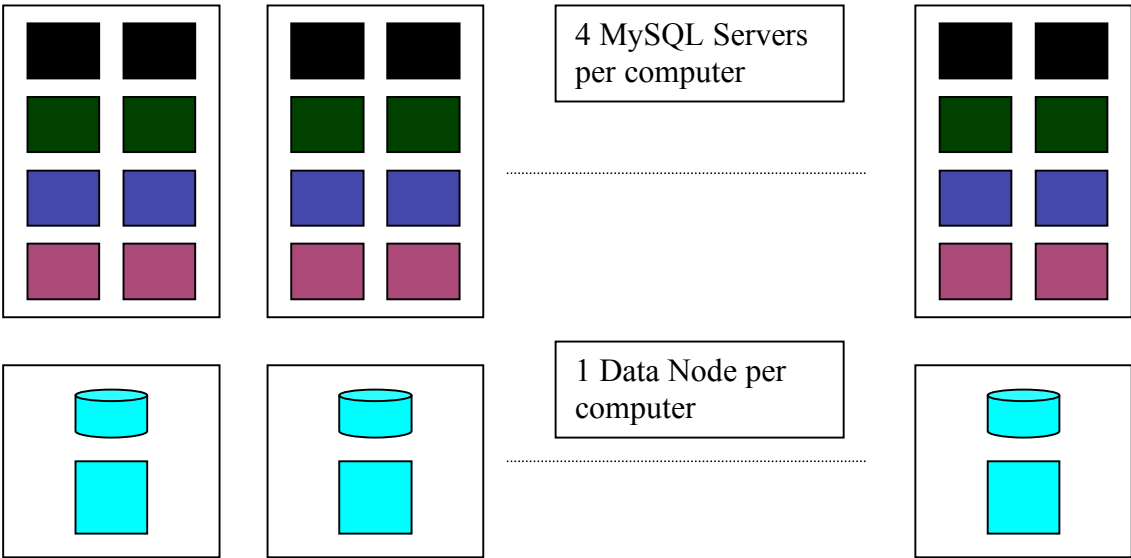
In the figure below we see a figure showing a preferred configuration in large clusters where data nodes are mostly there to provide more CPU resources. Then it's most efficient to use a balanced architecture with 4 nodes per quad core computer. It's the cheapest way to configure MySQL Cluster with maximal CPU resources in the data nodes. The architecture provided below assumes the use of Dolphin Express in the computers, otherwise only 3 data nodes per computer should be used.



In this configuration with only 2 data nodes per computer we can put in twice as much memory into the system. It is also easier to add more disks per data node, e.g. to handle two disks per data node with a RAID10 configuration.



And finally in the last figure we show the HW configuration that provides the most memory, disk and cluster interconnect resources per data node. In this case we have only one data node per computer and thus we can scale the cluster all the way up to 4 TB of memory and 1024 disks. For memory tables the disks are mostly needed for recovery operations but they can also be used for disk-based tables. We have not done any benchmarking on disk-based tables yet.



10 Benchmarking Results

We will show results of the DBT2 benchmark; this is an open source implementation of the TPC-C benchmark by OSDL (Open Source Development Labs). The TPC-C benchmark simulates the operations of a global corporation running a large number of warehouses. The runs of the DBT2 benchmark reported here doesn't conform to the TPC-C specification so should not be compared with other TPC-C results but they are useful to compare with the

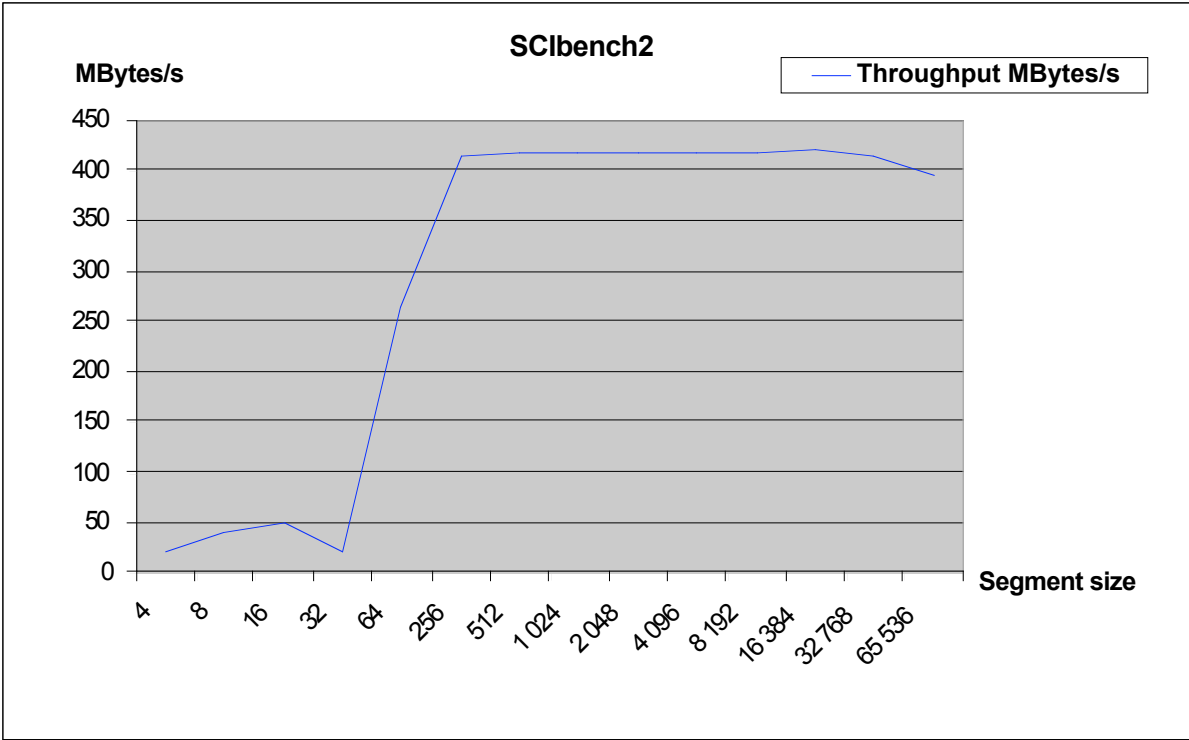
results as provided here from similar executions using Ethernet. A major diversion from the TPC-C specification used in these tests is that the think time for each terminal is zero to ensure that we can have as much parallelism per warehouse as possible.

10.1 Low-level Benchmarks

These benchmarks are provided to give an idea of the base benefits of the Dolphin Express. We start with scibench2 which is a low-level test that shows what is possible with the Dolphin hardware. The second test is NetPIPE, a standard cluster interconnect test that measures latency in sending messages of various sizes.

10.1.1 Results from scibench2

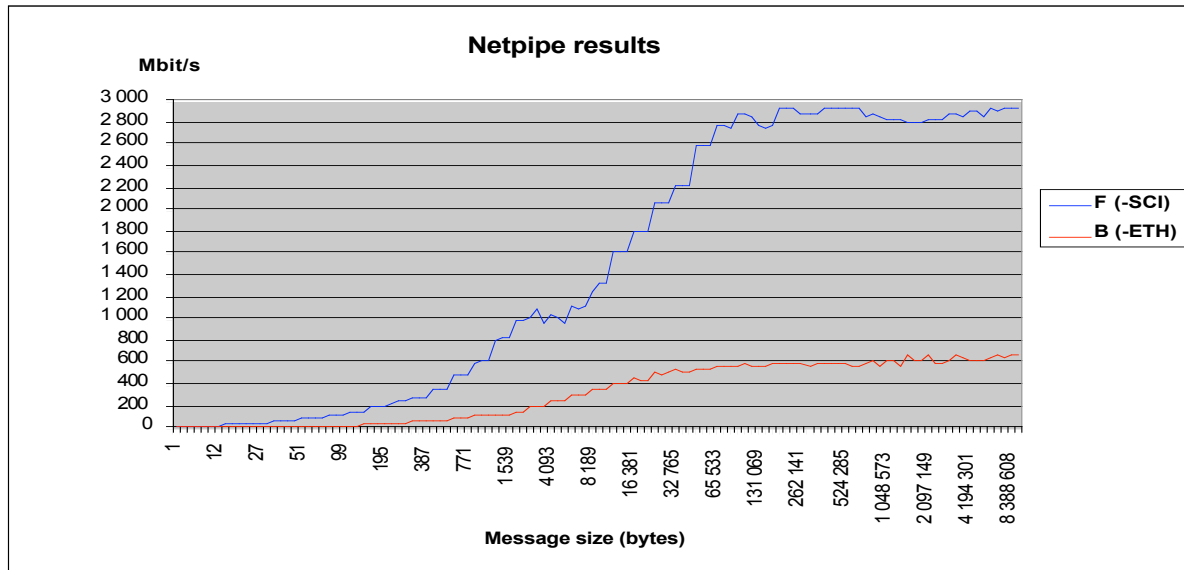
These tests are executed without waiting for response from receiver. Thus it tests how fast data can be sent from sender. As can be seen one comes up to the maximum speed of writes already at sizes of 256 bytes. The D350 card is able to get up to around 700 Mbytes per second but the hardware used in the test only had a PCI Express x8 slot which is wired as a x4 slot. This limits the bandwidth this computer can achieve.



10.1.2 Results from NetPIPE

When integrating Dolphin hardware with sockets it is necessary to also ensure validity of message, handle resends and so forth. The Dolphin hardware will detect failed sends through hardware integrated checksums. The software must however check hardware counters about this. As can be seen from the results there is a small fixed overhead when sending small messages. To achieve optimal cost of sending messages at all message the various protocols are used in a span where they are providing optimal response time.

It is also easy to see the benefits that Dolphin Express provides over Ethernet. This benefit comes both from the low latency and from the higher bandwidth of the Dolphin Hardware.



10.2 Computers used in the test

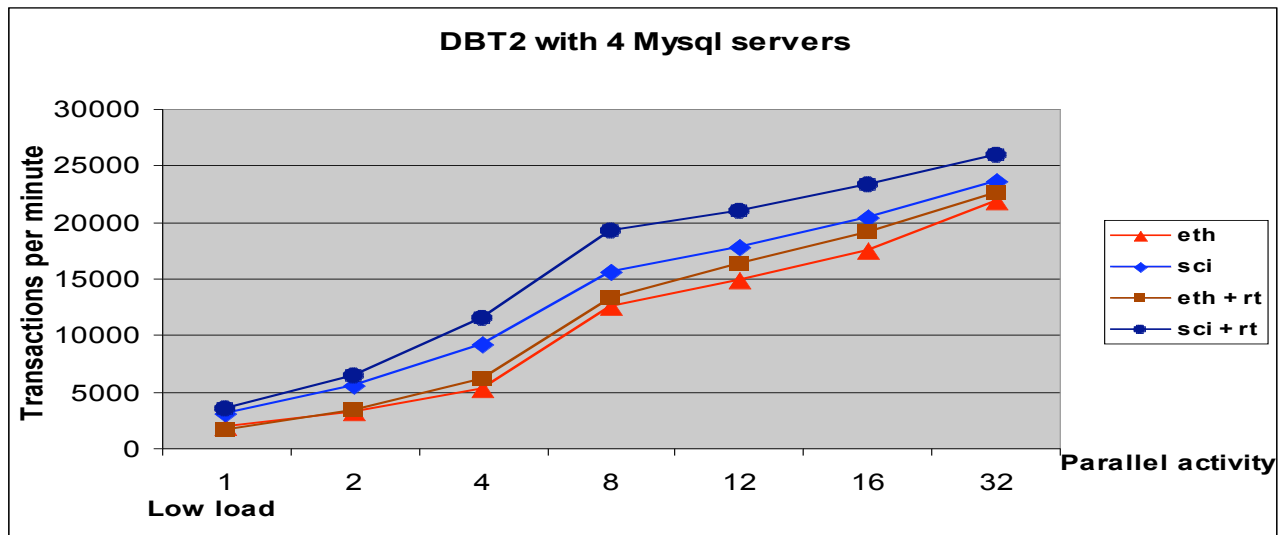
The benchmark configuration used in this test is based on Intel computers dual sockets and the Multi-Core Intel Xeon, the actual processors used in these tests were dual-core. There is between 2 and 9 such computers in each test. Each machine is equipped with dual CPU's (each CPU contains dual core CPU's @2.67 GHz). There is 4, 8 or 16 GBytes of memory in each machine. Each machine only had one disk, this lead us to perform some test runs without disk writing. To sustain disk writes from 4 data nodes on a quad-core machine it is necessary to have about 4 disks as well. Each machine has 2 gigabit Ethernet connections on the motherboard and is equipped with Dolphin Express cards.

We also did a number of similar tests using desktop computers based on the Pentium D processor. We saw roughly 75% improvement of performance using Dolphin Express and MySQL Cluster.

10.3 DBT2 results

10.3.1 2 Data Nodes and 4 MySQL Servers on 2 computers

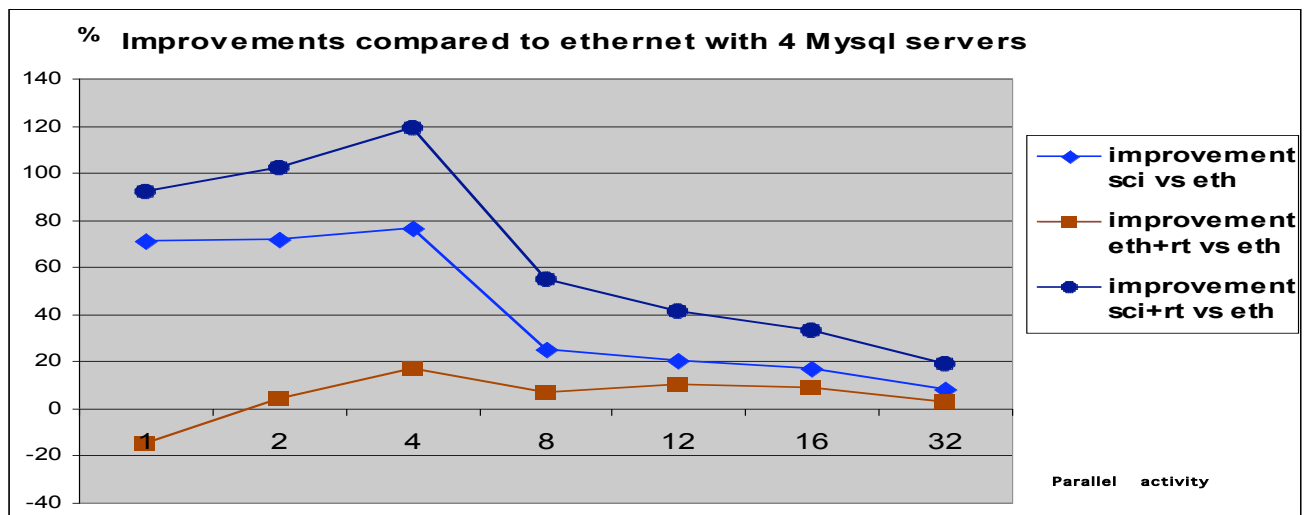
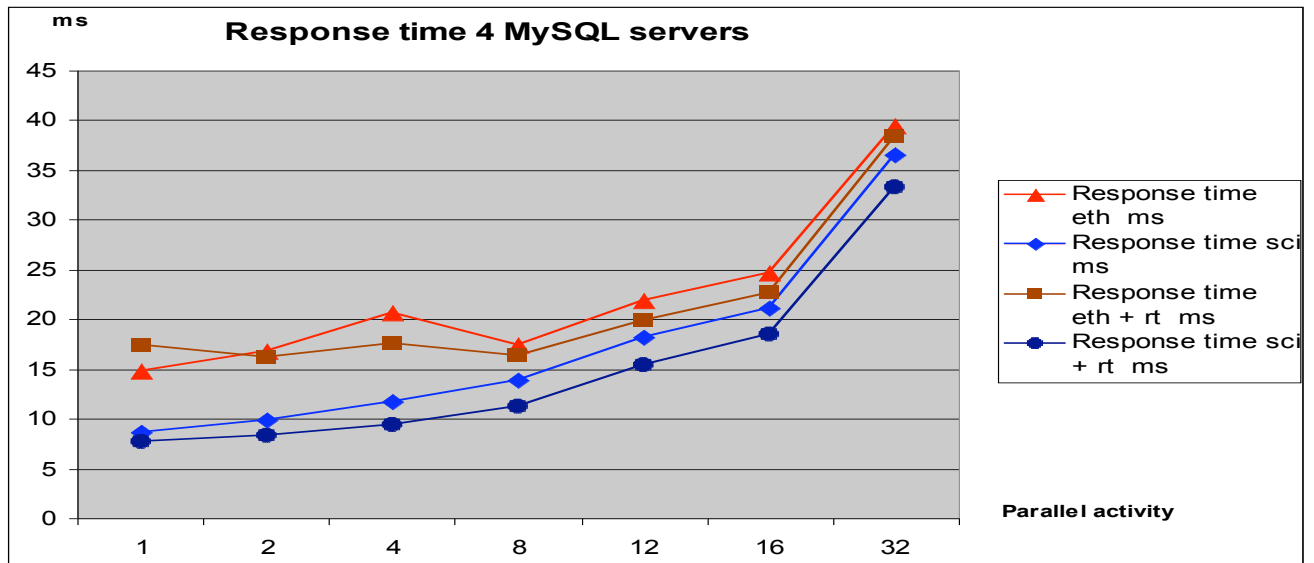
This configuration uses a version of MySQL Cluster which makes the application distribution aware, there are abundant CPU resources for the data nodes and also the cluster is very small. A small cluster in MySQL Cluster, at high load, doesn't use very much CPU resources for the communication part in most cases. Thus the benefit possible for a cluster interconnect at high load is fairly small. What we can see however is that Dolphin Express has a much lower latency when the number of parallel queries in the cluster is low. We can see as much as 150% improvement using the 'real-time' extensions of MySQL Cluster.



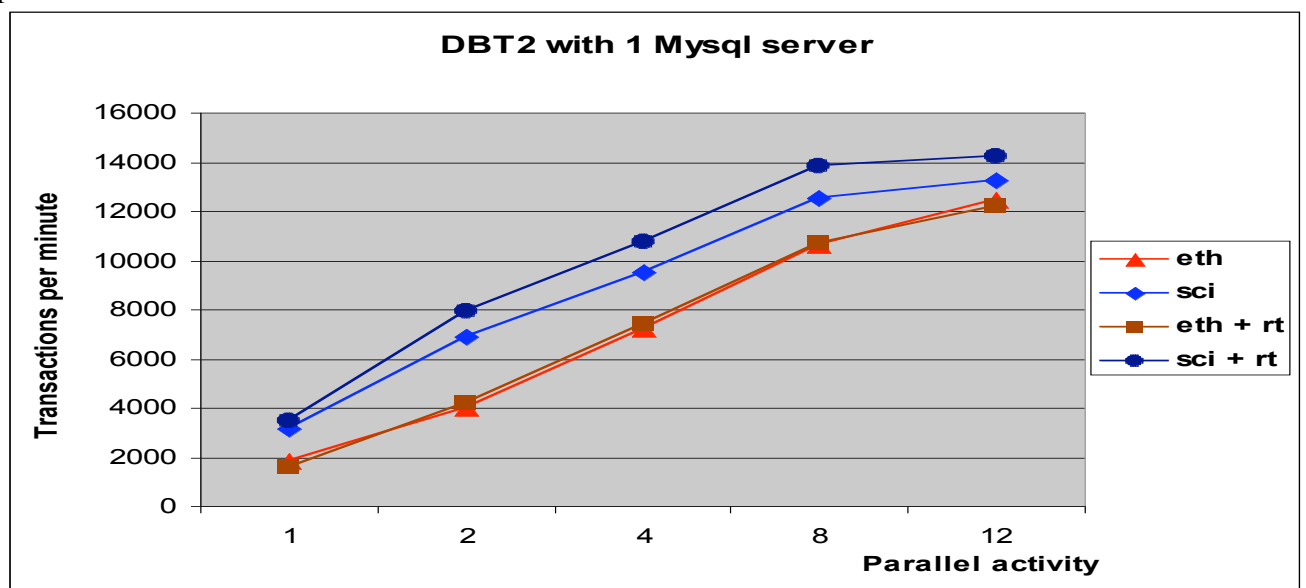
In these first graphs we show the performance of DBT2 using 4 MySQL Servers, the graph below shows the results in terms of improvements compared to the Ethernet results. The 'rt' refers to use of the 'real-time' extension of MySQL Cluster and 'sci' refers to using Dolphin Express (previously called SCI). This was accomplished by binding the main thread of the data nodes to one CPU and file system threads to another CPU, by setting SchedulerSpinTimer to 200 microseconds (thus never going to sleep before spinning for at least 200 microseconds) and SchedulerExecutionTimer to 50 microseconds (thus receiving more before sending if not at least 50 microseconds was spent executing). It was important to not bind CPU's to the same CPU as was used for Ethernet interrupt handling since this would have caused a drop of Ethernet performance of 20-30%.

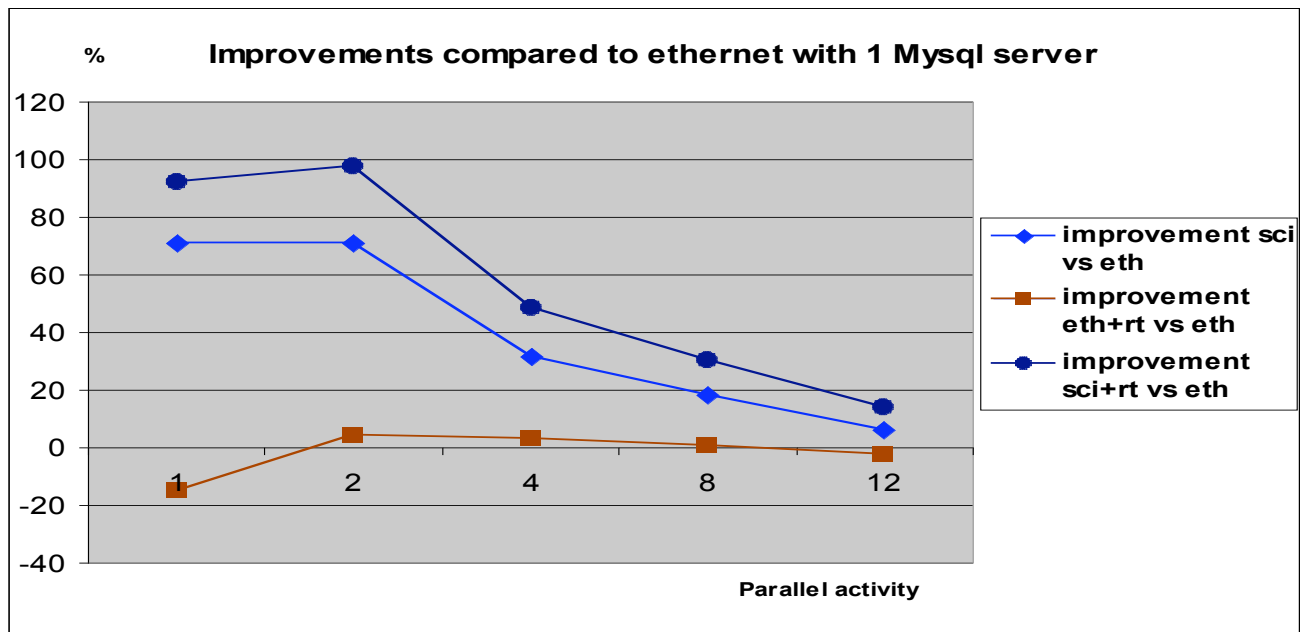
A few interesting things we find in the graphs is that Dolphin Express gets a higher benefit when there are several MySQL Servers using a single active thread. It is also obvious that having a second active thread in the MySQL actually improves the response time of both threads and in the Ethernet case even in a fairly significant manner. Thus Ethernet is much more susceptible to response time variance since it is so much in the need of concurrent active threads for best response time.

What we see is that up to 8 threads we can handle more transactions at a very good maintained response time. We improve throughput by 650% but only increasing response time by 50%. Whereas from 8 to 32 parallel activities we only increase throughput by 30% at the cost of a 200% increase in response time. Thus optimum for 'real-time' applications is here at 8 parallel activities where we can see that Dolphin Express using the 'real-time' extension have approximately a 50% performance advantage compared to Ethernet.

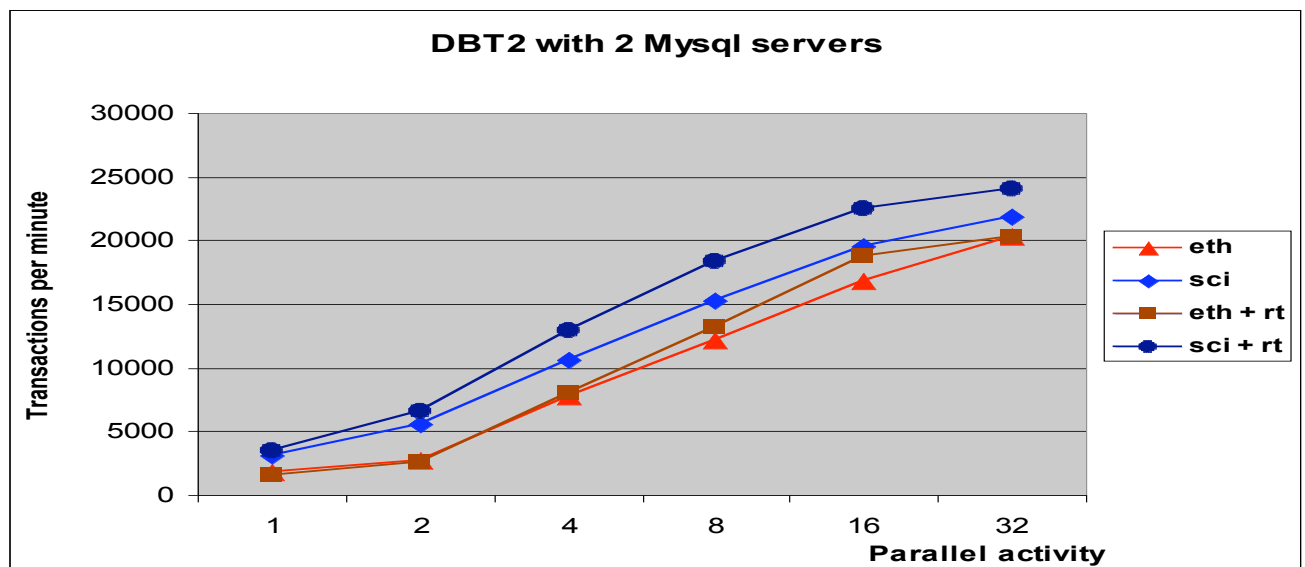


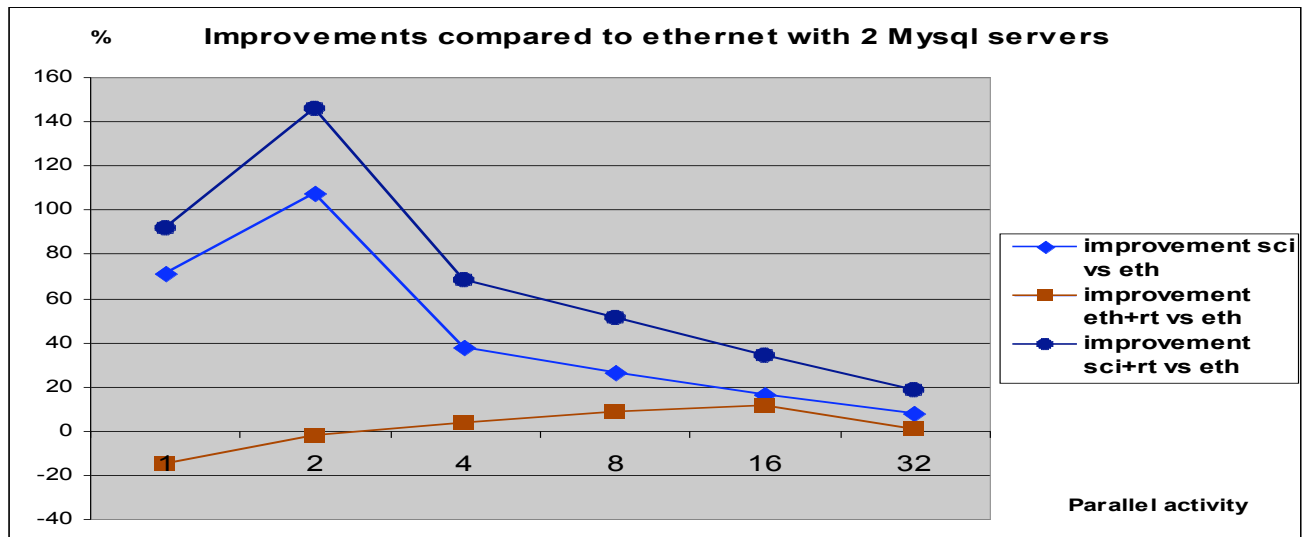
We'll also have a look at performance of a single MySQL Server. What we can see in the graphs below is that the results are similar; we still find that we scale fairly well up to 8 parallel activities.





Finally we show the same benchmark figures using 2 MySQL Servers. There is clearly a very large advantage to use a second MySQL Server; we get around 35% more throughput at 8 parallel activities. Going to 4 MySQL Server gives approximately 5% improvement thus notable but also requires more administration and hardly worth the effort. The benefit of Dolphin Express, using the 'real-time' extension at 8 parallel activities, stays around 50% independent of the number of MySQL Servers. We can also see that the 'real-time' extension for Ethernet has some variance in when it benefits. It mainly benefits when used where data nodes execute on their own machine with dual cores. In an earlier white paper we showed this benefit to be around 20% and even 25% in scenarios with lock contention.





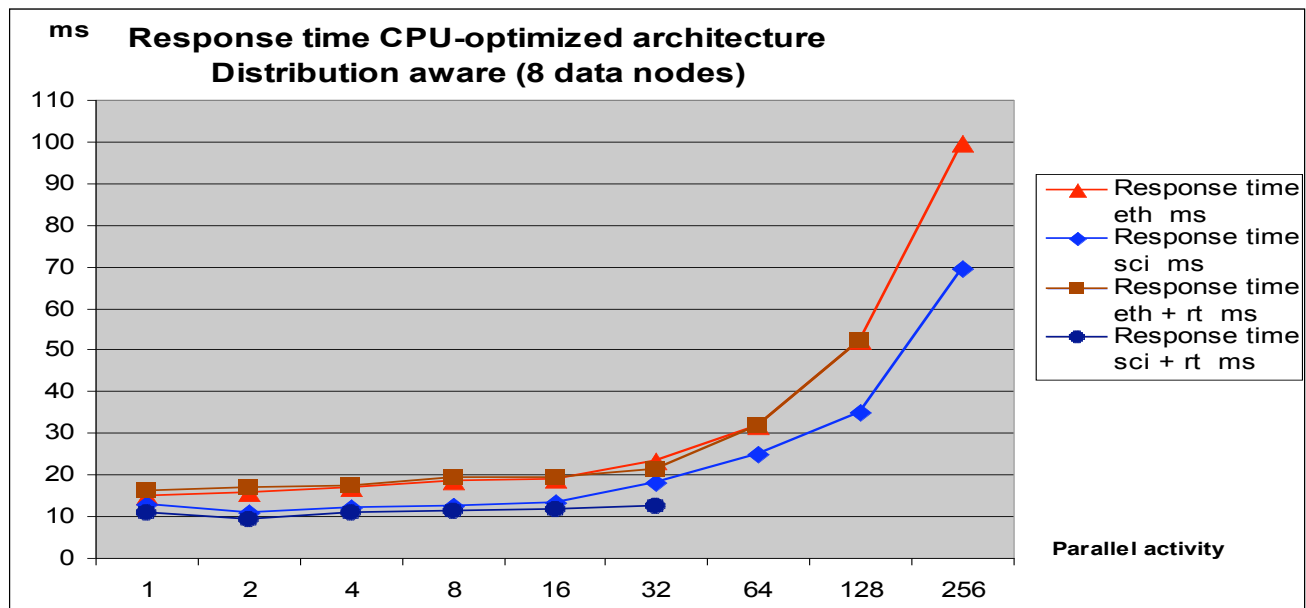
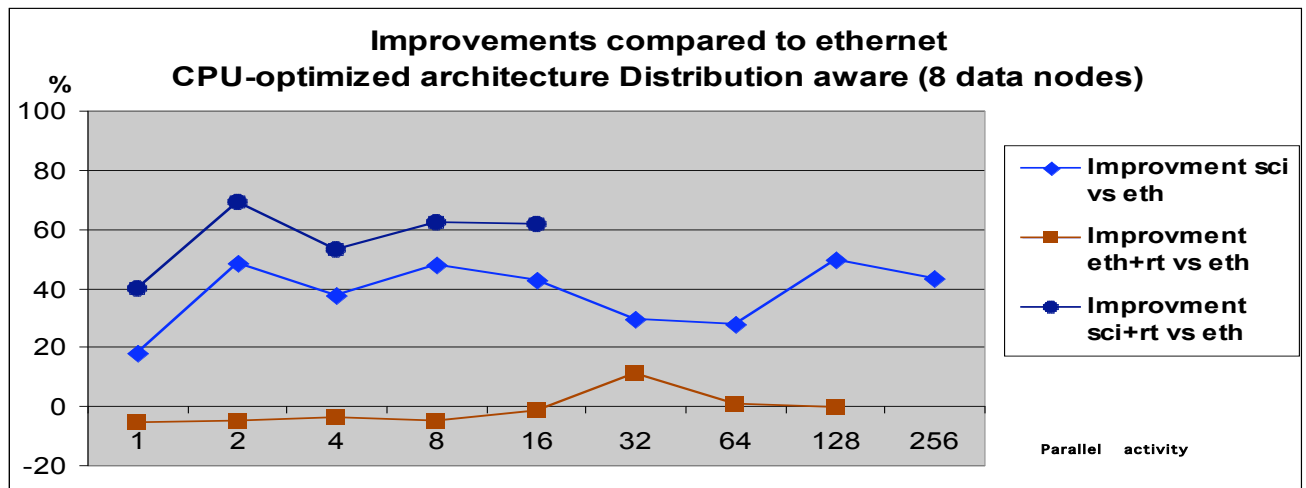
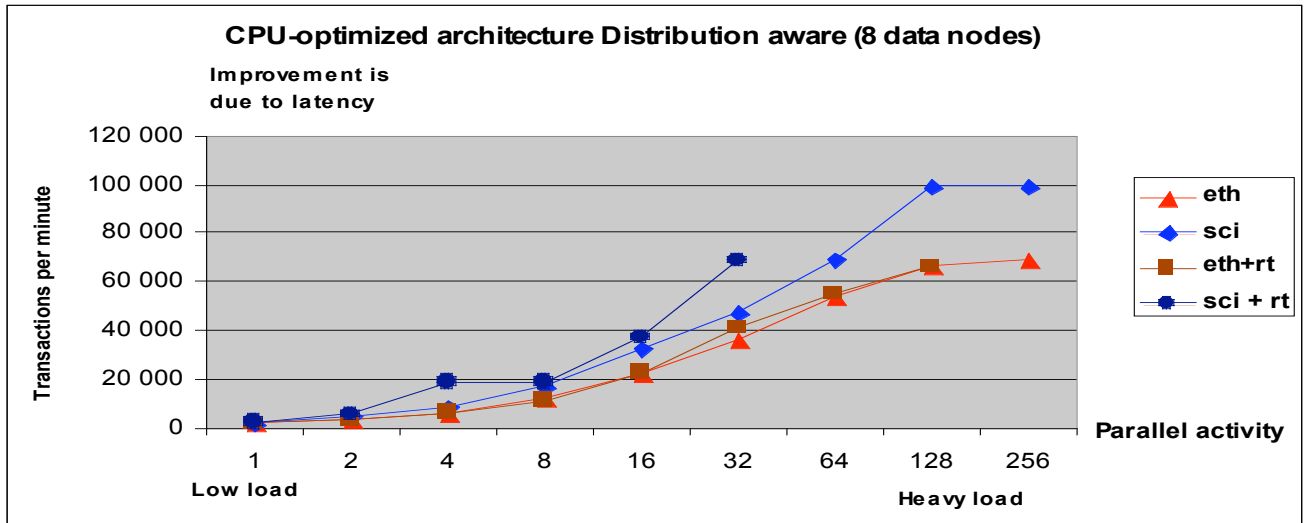
10.3.2 4 Data Nodes on 4 computers, 16 MySQL Servers on 4 computers

This benchmark used a version of the software which is not distribution aware so this is similar to the results one would get using version 5.0 of MySQL Cluster. Also this benchmark had an abundance of CPU resources since there was only one data node on a quad core machine. A data node requires about 1 core. A 4-node cluster is still a fairly small cluster so the effect of not being distribution aware is not that great as we can see here. The figures from the 2-node benchmark are to a great extent seen here as well. We can see that the latency at low parallelism is greatly improved, all the way up to 190% improvement and that as parallelism goes up the effect of communication improvements decrease.

What we see here is roughly that all the way up to 64 parallel activities we get 50% more throughput by doubling the number of parallel activities. At the point of 64 parallel activities the improvement of Dolphin Express is at 30%. In this benchmark we have no measurements using the 'real-time' extensions of MySQL Cluster. We expect this would improve numbers for Dolphin Express by around 20%.

The response time is fairly similar in this cluster as it was in the 2 data node cluster up until around 8 parallel concurrent activities. After that in this cluster the performance continues growing for a longer time. We also have more than twice the performance compared to the cluster with 2 data nodes. This is mostly due to the fact that we have a quad core computer per data node and thus no contention at all for CPU resources.

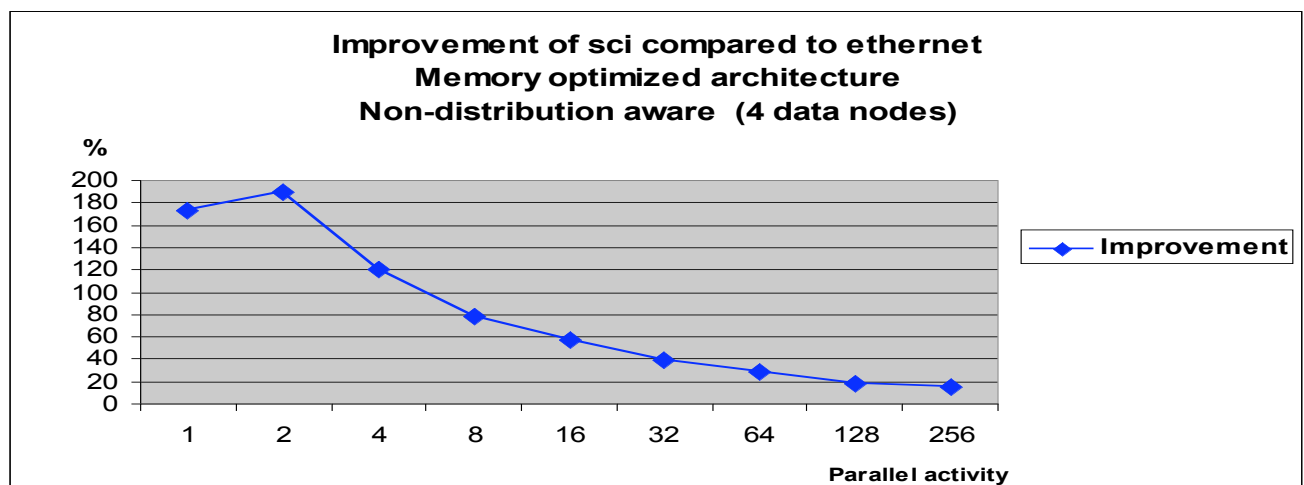
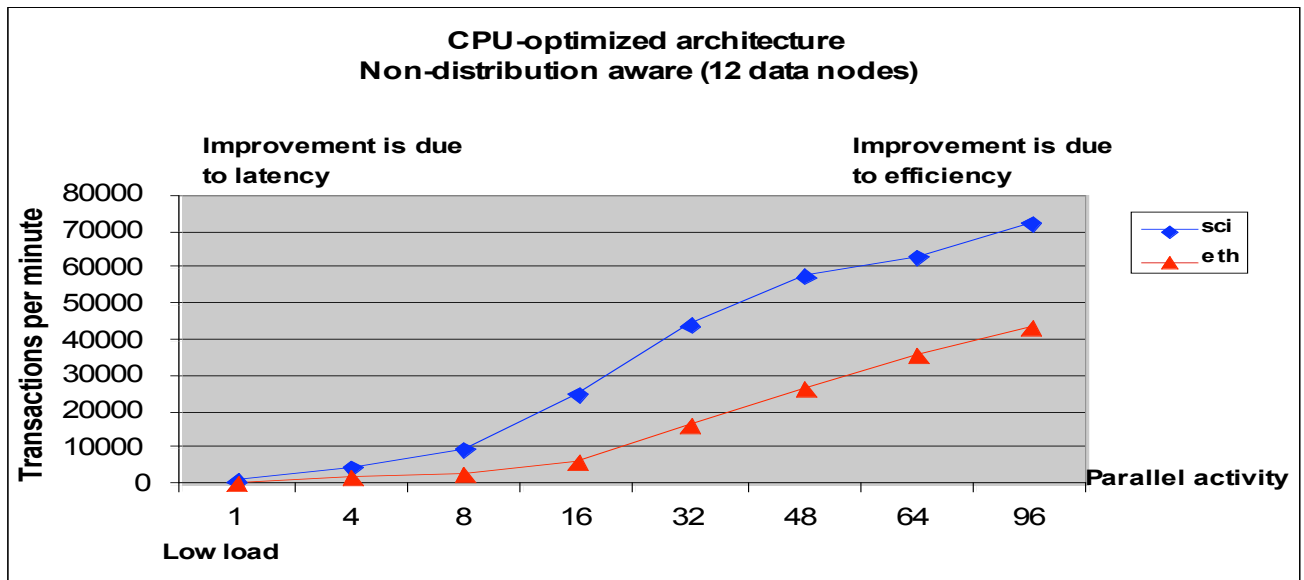
using the 'real-time' extension versus Ethernet. What we can see here is that we have roughly the same latency using 32 parallel activities here as we had using 8 parallel activities in a 2-node cluster. Thus MySQL Cluster has very good scalability for 'real-time' applications.

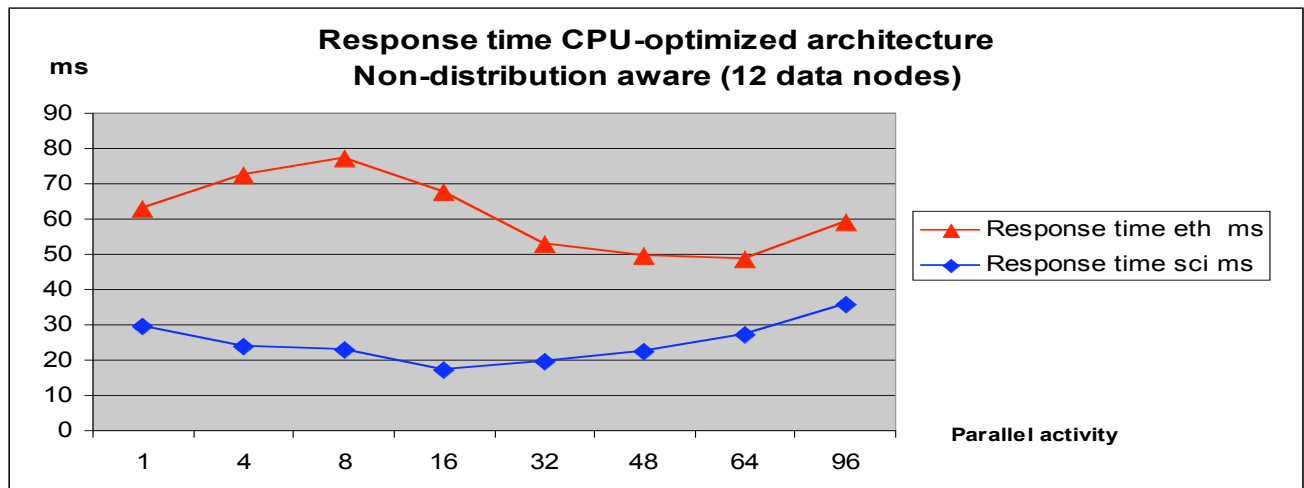


10.3.4 12 Data Nodes on 3 computers, 24 MySQL Servers on 6 computers

The final configuration is representing an application which is not distribution aware on a large cluster where CPU resources are limited as well. This is representing the really worst case possible for Ethernet whereas Dolphin Express still provides very good numbers. The improvement is all the way up to 292% and the minimum improvement is 66%. Thus Dolphin Express gives MySQL Cluster really good performance even in the presence of badly behaving applications. So this benchmark really shows the benefits in using Dolphin Express in that your application performance becomes much less susceptible to a bad design. This means that one can reach to a point where application performance is good enough much faster and thus improve delivery times, decrease development time and still improve performance of the application.

An interesting fact in these cluster configurations is that increasing parallelism actually improves response time up to 16 threads and that we have almost the same response time all the way up to 48 parallel activities.





11 Conclusions

This technical white paper has shown how MySQL Cluster on Multi-Core Intel Xeon using Dolphin Express can be used to build a 'real-time' DBMS product with very high performance and very high availability.

In most configurations we see around 50% improvement using Dolphin Express where we have optimum response time versus throughput. In some configurations the improvements were much more than this. Particularly as cluster size increases and/or distribution awareness of application decreases the benefits of Dolphin Express increases.

The major benefits that the user gets by using Dolphin Express are:

- Much shorter response time due to the very short latency of the Dolphin Express interconnect
- Higher bandwidth
- Decreased cost of socket interaction due to minimized interrupt processing
- Increased availability through much shorter node failover times
- Increased availability through very efficient Dolphin Express interconnect failover
- Increased throughput

The use of the Dual-Core Intel Xeon 5100 enabled a 75% improvement in performance overall and it also improved the scalability of the MySQL Server to twice as many threads per MySQL Server process. Many of the new MySQL Cluster Carrier Grade Edition features are enabled by the enhanced features Intel Core2 micro-architecture and the Multi-Core Intel Xeon processors.

In the MySQL documentation it is recommended to use Dolphin Express from 8 data nodes and upwards. In this paper we have verified with numbers this recommendation. We have also shown that Dolphin Express can greatly benefit also in 2 and 4 data node configurations and particularly so for 'real-time' applications. There are also many optimisations specifically designed for MySQL Cluster in the Dolphin Express driver which won't be found in other high-end cluster interconnects. In addition Dolphin has tested its driver extensively for use in MySQL Cluster.