




ORACLE®

MySQL Cluster – Performance Tuning

Johan Andersson

Principal Field Technologist





The presentation is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.



General Design Principles

- MySQL Cluster is designed for
 - Short transactions
 - Many parallel transactions
- Utilize Simple access patterns to fetch data
 - Solution that scales!
- Analyze what your most typical use cases are
 - optimize for those

Overall design goal

Minimize network roundtrips for your most important requests!



General Design Principles

- Application spaces where Cluster is being used heavily
 - Subscriber databases (telecom)
 - Session management
 - Online gaming
 - Finance
 - E-commerce
 - As a Shard catalog in web 2.0 shops
- The key denominator for all these applications are:
 - high throughput, low response times, high-availability, and simple access patterns.
- Reporting is typically performed on a subsystem
 - Replicate to a slave (e.g, MYISAM or INNODB database)



Tuning Options

Schema Optimization

- De-normalization
- Optimize data types

Query Tuning

- Batching
- Rewrite slow queries
- Index Tuning

Parameter Tuning

- Use a good Configuration (affects mostly stability)
- Mainly MySQL server parameters

Network / OS Tuning

- Tune Network (TCP) buffers (not the scope of this presentation)
- Cluster Interconnects

Hardware Tuning

- Faster CPU/Disk (not the scope of this presentation)

Detecting Problems – PT 101

- Enable the slow query log!

- `set global slow_query_log=1;`
- `set global long_query_time=3; //3 seconds`
- `set global log_queries_not_using_indexes=1;`
- **Slow queries will be written in the slow query log:**

```
mysql> show global variables like 'slow_query_log_file';
+-----+-----+
| Variable_name      | Value                                |
+-----+-----+
| slow_query_log_file | /data1/mysql/mysqld-slow.log      |
+-----+-----+
1 row in set (0.00 sec)
```

- **Slow Queries will be written in plain text.**

- Or use MEM (but MEM cannot monitor data nodes)



Detecting Problems – PT 101

BEGIN

1. Start by analyzing the slow query log
Change `long_query_time` if needed
2. Use `EXPLAIN` to figure out if the query is
 - Using the correct indexes
 - JOINing the tables in the wrong order
 - so bad it needs to be rewritten.
3. Re-run the optimized typical use cases using `mysqlslap`
4. GOTO BEGIN;

END;

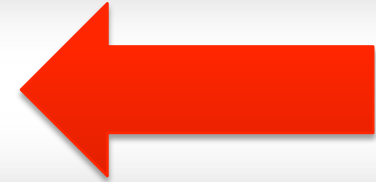
- Other tools such as `mysqlsla` can also be used
- Performance tuning is a never-ending task.
- Never tune unless you can measure and test

Don't optimize unless you have a problem

Tuning Options

Schema Optimization

- De-normalization
- Optimize data types



Query Tuning

- Batching
- Rewrite slow queries
- Index Tuning

Parameter Tuning

- Use a good Configuration (affects mostly stability)
- Mainly MySQL server parameters

Network / OS Tuning

- Tune Network (TCP) buffers (not the scope of this presentation)
- Cluster Interconnects

Hardware Tuning

- Faster CPU/Disk (not the scope of this presentation)



Schema Optimization - Data Types

- Denormalize tables
 - Tables having the same `PRIMARY KEY` can be denormalized
- Change Data Types
 - Does an `EMAIL` need to be a `TEXT`?

Schema Optimization - Denormalization

- Two tables with the same PRIMARY KEY can be denormalized into a single table:

USERID	VOIP_DATA
1	<data>
2	<data>
3	<data>
4	<data>

•USER_SVC_VOIP

USERID	BB_DATA
1	<data>
2	<data>
3	<data>
4	<data>

•USER_SVC_BROADBAND

- Requires two roundtrips to get data
- Denormalize:

USERID	VOIP_DATA	BB_DATA
1	<data>	<data>
2	<data>	<data>
3	<data>	<data>
4	<data>	<data>

•USER_SVC_VOIP_BB



Schema Optimization - Denormalization

- Normalized:

- ```
SELECT * FROM
 USER_SVC_BROADBAND AS bb, USER_SVC_VOIP AS
 voip
 WHERE bb.id=voip.id AND bb.id=1;
```
- **Total throughput = 12623.09 tps**
- Average response time=658us

- Denormalized:

- ```
SELECT * FROM USER_SVC_VOIP_BB AS bb_voip  
  WHERE bb_voip=1;
```
- **Total throughput = 21591.64 tps**
- Average response time=371us



Schema Optimization – Data Types

- BLOB/TEXT columns are stored in an external hidden table.
 - First 255B are stored inline in main table
 - Reading a BLOB/TEXT requires two reads
 - Read without lock will be upgraded to shared lock!
- Reading/Writing a VARCHAR/VARBINARY is less expensive.
- Change to VARBINARY/VARCHAR if:
 - Your BLOBs/TEXTs can fit within an 8052B record (and you need a 4B PK as well)
 - (record size is currently 8052 Bytes)



Schema Optimization – Data Types

- `SELECT data1, data2 FROM t1 WHERE id=<rand>`
 - `sizeof(data1) = 1024B, sizeof(data2) = 1024B.`
- 1 App - 8 Threads , 1 MySQLD, 2 Data nodes
- data1 and data2 represented as `BLOBs`
 - **5844 TPS**
- data1 and data2 represented as `VARBINARYs`
 - **19206 TPS**
- **Note 1:** BLOB/TEXT are also more expensive in Innodb as BLOB/TEXT data is not inlined with the table. Thus, two disk seeks are needed to read a BLOB.
- **Note 2:** We recommend (for any storage engine) to store images, movies etc outside the database on the filesystem.

Schema Optimization - PK selection

- Engineer your schema for the problem you need to solve!
 - Call setup? Locate all friends of a user?

- Very common...

PK

UNIQUE KEY

<u>ID (auto_inc)</u>	USER_ID	FRIEND_ID
1	10001	11000
2	10001	11001
3	10001	11002
4	10002	12022

- Better:
 - Introduce PK <USER_ID, FRIEND_ID>
 - Get rid of column ID
 - Get rid of the UNIQUE (as it is now the PK)

Tuning Options

Schema Optimization	<ul style="list-style-type: none">• De-normalization• Optimize data types
Query Tuning	<ul style="list-style-type: none">• Batching• Rewrite slow queries• Index Tuning
Parameter Tuning	<ul style="list-style-type: none">• Use a good Configuration (affects mostly stability)• Mainly MySQL server parameters
Network / OS Tuning	<ul style="list-style-type: none">• Tune Network (TCP) buffers (not the scope of this presentation)• Cluster Interconnects
Hardware Tuning	<ul style="list-style-type: none">• Faster CPU/Disk (not the scope of this presentation)





Simple Access Patterns

- Simple Access Patterns are key to build scalable and high performing solutions (this is not subject to Cluster only)
 - PRIMARY KEY lookups are done in constant time $O(1)$
 - Fastest way to access data in MySQL Cluster
 - INDEX searches are done in $O(\log n)$ time.
 - JOINS are ok if you understand what can make them slow.
 - If your most important requests are 10-way JOINS with huge result sets then Cluster may not be for you.
 - Or use scale out (write to cluster read from innodb): <http://johanandersson.blogspot.com/2009/05/ha-mysql-write-scaling-using-cluster-to.html>

Operation Cost

- Cost of typical operations (depends on HW/Network)

PK	Avg cost (us)	Min cost (us)	Normalized (scalar)
Insert 4B + 255B	826	600	2.82
read 255B	293	174	1
update 255B	697	505	2.38
delete	636	202	2.17
Index/FT			
1 record (index)	694	400	2.37
2M FT Scan	4.71(SEC)	-	-

- Synchronous replication adds ~2.1x - 2.8x for writes compared to reads
- Index scan takes 2.4x longer than PK read
- Test was with 8 threads connecting to one mysqld
- 'bencher' was used to generate the load. (Xeon 5160 @ 3.00GHz)



Batching

- MySQL Cluster allows batching on
 - INSERT (PK)
 - Most PK UPDATE
 - DELETE (PK)
 - SELECT (PK and some INDEX scans and not in JOINS)
- Batching means
 - One transaction with >1 operation are executed in one round-trip



Batching

- Example – Insert 1M records
 - No batching:
 - `INSERT INTO t1(data) VALUES (<data>);`
 - Batching (batches of 16):
 - `INSERT INTO t1(<columns>) VALUES (<data0>), (<data1>) ..., (<data15>)`
 - 50 seconds to insert 1M records
 - **15 times faster!**



Batching

- Read 10 records services for a user:
 - PK is <userid, friend_id>
- Batching (batches of 10):
 - `SELECT * FROM t1 WHERE user_id=1 AND friend_id IN (1,2,3,4,5,7,8,9,10);`
 - **0.001s**
- No batching:
 - `10 x SELECT * FROM t1 WHERE user_id=1 AND friend_id={ id };`
 - **0.006s**

Batching

- Another way – batching on different tables

```
SET transaction_allow_batching=1; /set on the  
connection
```

```
BEGIN;
```

```
INSERT INTO user(uid, fname, lname, email) VALUES  
  ( ... );
```

```
10 x INSERT INTO service(uid, sid, data ) VALUES  
  ( ... );
```

```
COMMIT;
```

- The above will be executed in one batch (one roundtrip)
 - `transaction_allow_batching=0`: 1223 TPS
 - `transaction_allow_batching=1`: **2204 TPS (80% faster)**
- Batching using `transaction_allow_batching` does not work with
 - UPDATE .. SET X=X+1 .. , JOINS, REPLACE

Efficient Scanning – Partition Pruning

- Scanning only one partition is *sometimes* better than scanning all partitions (all nodes).
 - By default, all index scans hit all data nodes – good if big result set.
 - User-defined partitioning can help to improve equality index scans on part of a primary key.
 - ```
CREATE TABLE user_friends (user_id,
 friend_id,
 data,
 PRIMARY KEY(user_id, friend_id))
PARTITION BY KEY(user_id);
```

    - All data belonging to a particular `user_id` will be on the same partition.
  - ```
SELECT * FROM user_friends WHERE user_id=1;
```

 - Only one data node will be scanned (no matter how many nodes you have)



Efficient Scanning – Partition Pruning

- You can verify if you got it correct checking the Ndb_pruned_scan_count status variable
 - Increases when a pruned scan occurs

```
mysql> select * from user_friend where user_id=1;
```

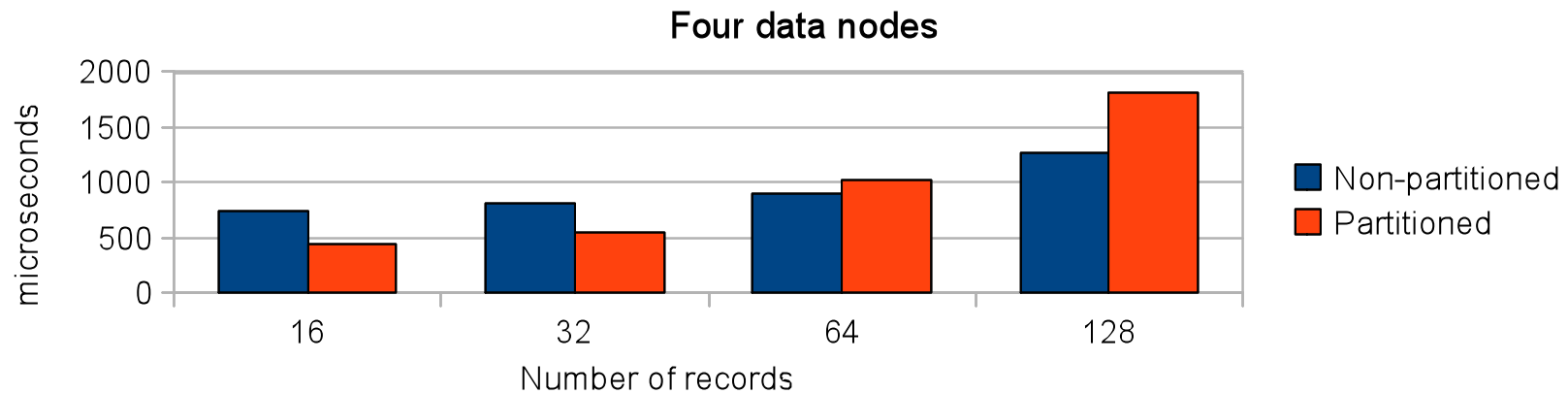
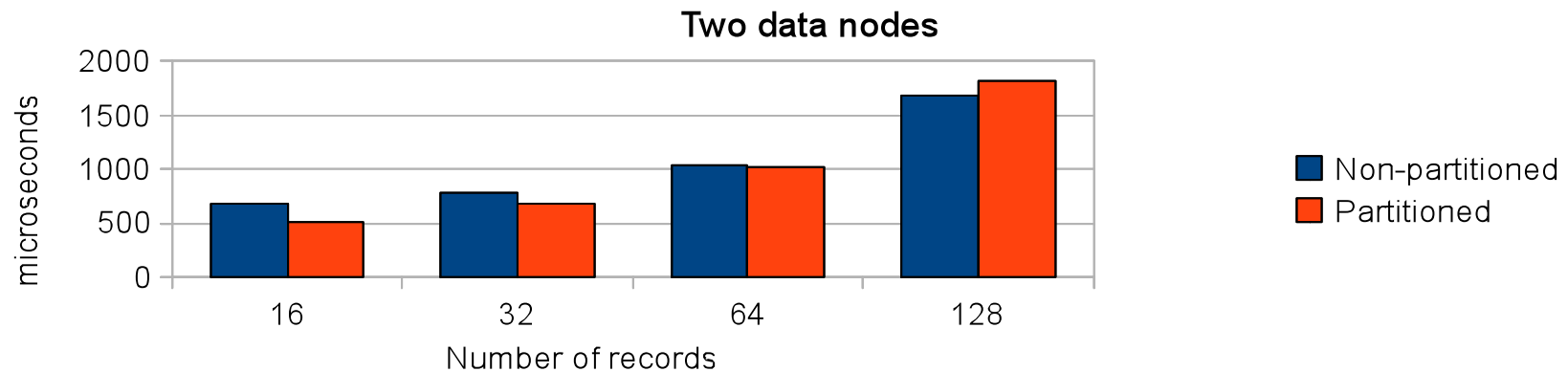
```
mysql> show global status like  
      'ndb_pruned_scan_count';
```

```
+-----+-----+  
| Ndb_pruned_scan_count | 1      |  
+-----+-----+
```

```
1 row in set (0.00 sec)
```

Efficient Scanning – Partition Pruning

- Partition Pruning is better up to a certain point
 - Depends on number of data nodes and records retrieved



Query Optimization – JOINS

- JOINS are executed in the MySQL server.
- The OPTIMIZER in MYSQL only knows one algorithm
 - Nested Loop Join
 - This algorithm is not brilliant in its effectiveness
- If we have the following query:
 - `SELECT fname, lname, title
FROM a,b
WHERE b.id=a.id AND a.country='France';`

Authid (PK)	Frame	Iname	Country
1	Albert	Camus	France
2	Sully	Prudhomme	France
3	Johann	Goethe	Germany
4	Junichiro	Tanizaki	Japan

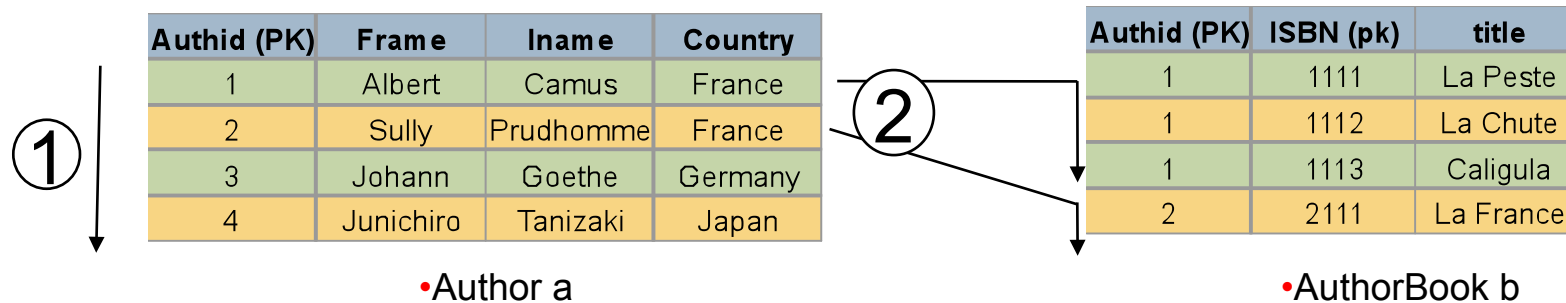
•Author a

Authid (PK)	ISBN (pk)	title
1	1111	La Peste
1	1112	La Chute
1	1113	Caligula
2	2111	La France

•AuthorBook b

Query Optimization - JOINS

- `SELECT fname, lname, title FROM a,b WHERE b.id=a.id AND a.country='France';`



- 1. Index scan left table to find matches.
- 2. For each match in 'a', find matches in 'b'
 - In this an index scan on the right table on b.id for each matching record in 'a'
 - This could be very expensive if there are many records matching `a.country='France'`



Query Optimization - JOIN

- The main performance limiter for JOINS are
 - Number of tables in JOIN
 - Number of Records matching the JOIN criteria
- In general
 - JOINS are limiting scalability even for INNODB/MYISAM
 - It is a complex access pattern
 - JOINS should be as simple as possible
 - WHERE – conditions should be as limiting as possible
 - Consider this:
 - **Every inspected record costs about 200us for a PK join**
 - A join hitting 2000 (2000 x 200 us) records → 0.4seconds
 - A join hitting 2 000 000 records → 40 seconds
 - Using SCI/DX can help a lot as JOINS are subject to network latency that is problematic.



Query Optimization - JOIN

- Make sure the tables are joined in the correct order
 - Check with `EXPLAIN!`
 - Sometime the order is screwed up
 - Make sure you have the necessary indexes
 - Make sure tables are JOINed with the table having the best/most conditions comes first in the JOIN.
 - Preferably take the smallest table first in the JOIN
 - `STRAIGHT_JOIN` can help a lot



Query Optimization - SUB-SELECT

- Rewrite SUB-SELECTS as JOINS

- `SELECT x FROM t1 WHERE t1.id IN
(SELECT t2.id FROM t2 WHERE t2.y>10`

Becomes

- `SELECT x FROM t1,t2 WHERE t1.id=t2.id AND
t2.y>10;`

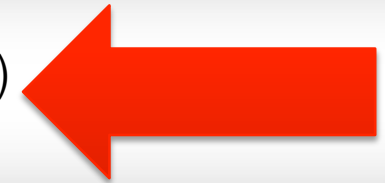


Indexes

- Don't trust the OPTIMIZER!
 - Statistics gathering is very bad
 - Optimizer thinks there are only 10 rows to examine in each table!
- If you have two similar indexes on a table
 - `index(a)`
 - `index(a,ts)`
- Use `FORCE INDEX` to use the correct index
 - Don't use `USE INDEX`
- Check with `EXPLAIN!`

Tuning Options

Schema Optimization	<ul style="list-style-type: none">• De-normalization• Optimize data types
Query Tuning	<ul style="list-style-type: none">• Batching• Rewrite slow queries• Index Tuning
Parameter Tuning	<ul style="list-style-type: none">• Use a good Configuration (affects mostly stability)• Mainly MySQL server parameters
Network / OS Tuning	<ul style="list-style-type: none">• Tune Network (TCP) buffers (not the scope of this presentation)• Cluster Interconnects
Hardware Tuning	<ul style="list-style-type: none">• Faster CPU/Disk (not the scope of this presentation)

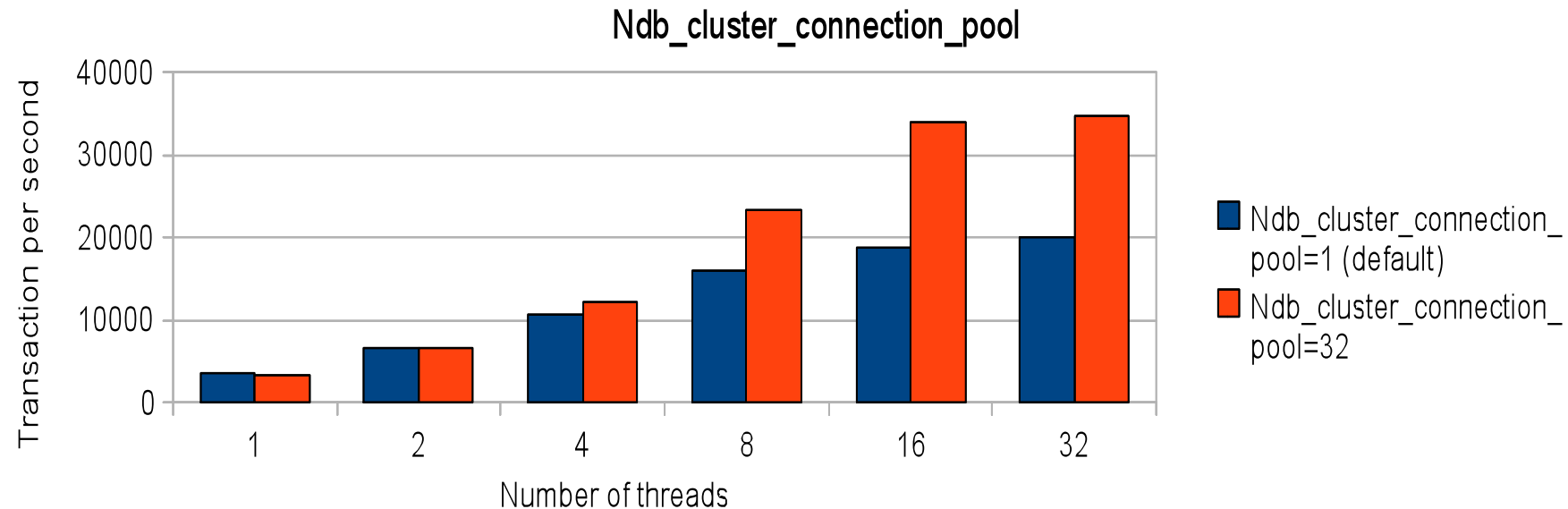




Parameter Tuning – `ndb_cluster_connection_pool`

- Problem:
 - A mutex on the connection from the mysqld to the data nodes prevents scalability.
 - Many threads → contention on the mutex
 - Must have many mysqld processes running...
- Solution:
 - `Ndb_cluster_connection_pool` (in `my.cnf`) creates more connections from one mysqld to the data nodes
 - One free `[mysqld]` slot is required in `config.ini` for each connection.
 - Threads load balance on the connections → less contention on mutex → increased scalability

Parameter Tuning – Ndb_cluster_connection_pool



- >70% better perf
- `Ndb_cluster_connection_pool=2x<CPU cores>` is a good starting point.
- www.severalnines.com/config allows you to specify this

Parameter Tuning – auto_increments

- A range of `auto_increments` are cached in the MySQL Server
 - ServerA gets 1..1024 , serverB gets 1025-2048
 - When out of values in range → go to data nodes, lock, fetch next range, unlock → serialization!
 - `ndb_autoincrement_prefetch_sz=1` (default - too small)
 - Must fetch new ranges all the time from data nodes! Round-trip!
- 16 BATCHED INSERTS / 8 THREADS / 1 APP
 - Default=1: 1211.91TPS
 - 256: 3471.71TPS
 - 1024 : 3659.52TPS
- Increase `ndb_auto_increment_prefetch_sz` depending on INSERT load.



Parameter Tuning - Misc

- Don't forget to set:
 - `thread_cache_size = <max_connections>`
 - `table_open_cache=512`
- Use `SHOW GLOBAL STATUS;`
 - If `Threads_created` increases -> increase `thread_cache_size`!
 - If `Opened_tables` increases -> increase `table_open_cache`!
- Please note that www.severalnines.com/config sets great default values! (the best in the industry actually)

Tuning Options

Schema Optimization

- De-normalization
- Optimize data types

Query Tuning

- Batching
- Rewrite slow queries
- Index Tuning

Parameter Tuning

- Use a good Configuration (affects mostly stability)
- Mainly MySQL server parameters

Network / OS Tuning

- Tune Network (TCP) buffers
(not the scope of this presentation)
- Cluster Interconnects

Hardware Tuning

- Faster CPU/Disk (not the scope of this presentation)





Network – Cluster Interconnects

- Cluster Interconnects
 - Instead of spending \$\$\$ on application tuning/development
 - Also great for DRBD!
- DX (SCI) is a Cluster Interconnect offering:
 - High Bandwidth (20 Gb/sec full duplex), Low latency (<2us)
 - Offers a socket interface – any socket based application benefits from it
 - 10 Gig-E form factor on cabling
 - Seamless fallback to Gig-E
 - **>2x more performance just plugging it in.**
 - DXH510 PCI Express Host Adapter (600USD list price Oct 2008) DXS410 DX 10 Port Switch (4200USD list price Oct 2008)
 - Read more at <http://www.dolphinics.com>



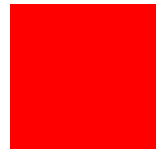
Other things

- Make sure you ***never***:
 - Run in SWAP – the data nodes will be sub-performing and you will have an unstable system.
- Make sure you ***do***:
 - Lock data nodes threads to CPUs not handling interrupts for ETH.
 - Vm.swappiness=0
 - Mount with noatime
 - On SUN CMT (T5240 etc) it is very important to create processor sets and bind interrupt handling to particular Core (HW Thread)



Tools

- Third party tools at www.severalnines.com
 - Configurator
 - Uses best practices for setting up a good config.ini and my.cnf
 - Scripts to control cluster from one single location.
 - CMON – Cluster Monitor
 - Monitor X number of Clusters (and mysqld statistics) from a single web interface
 - Sizer – Capacity Planning
 - Sandbox – Development package (localhost) for Cluster



Questions?

THANK YOU!

Johan.Andersson@sun.com

www.severalnines.com

johanandersson.blogspot.com