BENCHMARKING DATABASES

"An investigation of the TPC-H benchmark suite and techniques used in the performance optimization of Decision Support Systems (DSS)."

Case study: Microsoft SQL Server 2008, 64 bit, Enterprise Edition

Samy Kabangu

Supervisor: John Ebden

What is current?

- Large databases to be queried to answer business questions, or in data mining, to support decision in any application of need.

  eg: Business intelligence, OLAP

- The larger the database the longer it takes to retrieve data from it.

- Taking hours or even days of execution run !!!

Statement of the problem

1. An investigation of the of the TPC-H benchmark suite as a decision support workload.

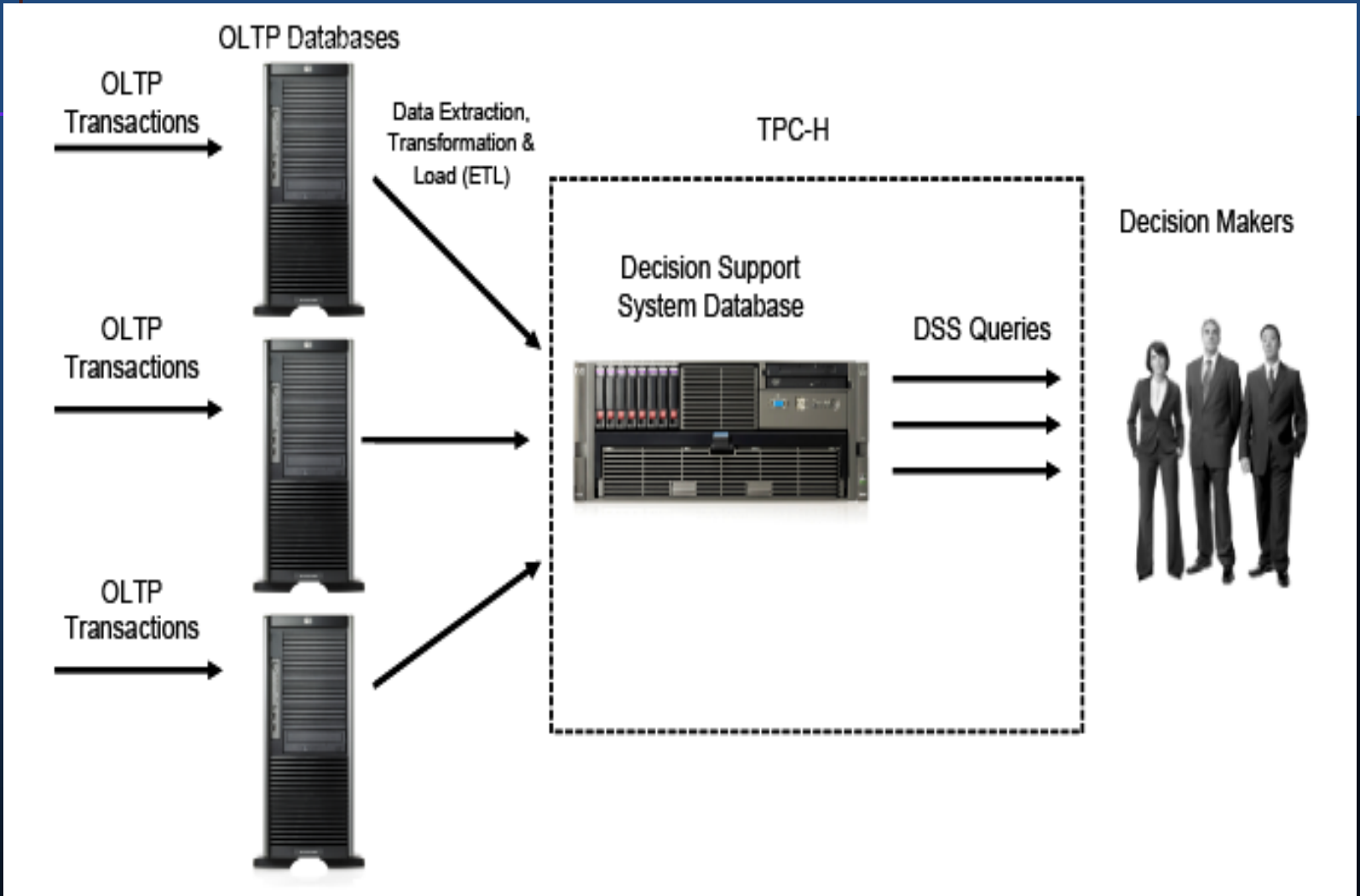
2. Exploration of different techniques used for optimizing decision support systems type of workload

What is the TPC-H

- The Transaction Processing Performance Council (TPC) benchmark suite for Decision support systems.

- With a workload consisting of 22 business oriented queries and different sizes of database depending on the Scale factor. (1GB, 10GB, 30GB, 100GB, 300GB, 3000GB, 10000GB)

TPC-H Operations



Methodology

- First task : Gathering information about the TPC-H
- Second task: Exploring techniques used in optimizing the performance of decision support workloads.
- Third: To optimize our custom application on which the TPC-H workload will be run based on the information gathered in the first and second task as well the performance results collected during the experimentation process.

Task 1 –The TPC-H (Background)

- The twenty two TPC-H queries are complex (involving joins, sub-queries, aggregate operations, nested loop and **Long running queries ; Data intensive high input/output disk activities**)

- It is designed in the third normal (schema)

- Different database size depending on the scale factor(1BG, 10GB, 30GB, 100GB, 300GB, 1000GB, 3000GB, 10000GB)

- The smallest scale factor for published results: 100GB

TPC-H TABLE ’S CARDINALITY

	Cardinality(rows)		
Tables	*Scale Factor 1*	*Scale Factor 10*	*Scale Factor 30*
PART	200000	2000000	6000000
PARTSUPP	800000	800000	24000000
LINEITEM	6000000	60000000	1800000000
SUPPLIER	10000	100000	300000
CUSTOMER	150000	1500000	4500000
ORDERS	1500000	15000000	45000000
NATION	25	25	25
REGION	5	5	5

Task 2 - Decision support Systems optimization techniques

Techniques experimented:

- Use of Indexes
- Query Parallel Processing (Take advantage of the multi-processor architecture platform used)

Techniques thought of (but not experimented):

- Use of RAID(Software or hardware)
- PC Clusters (distributed systems)

TASK 3- The performance optimization

Design consideration:

Hardware Platform:

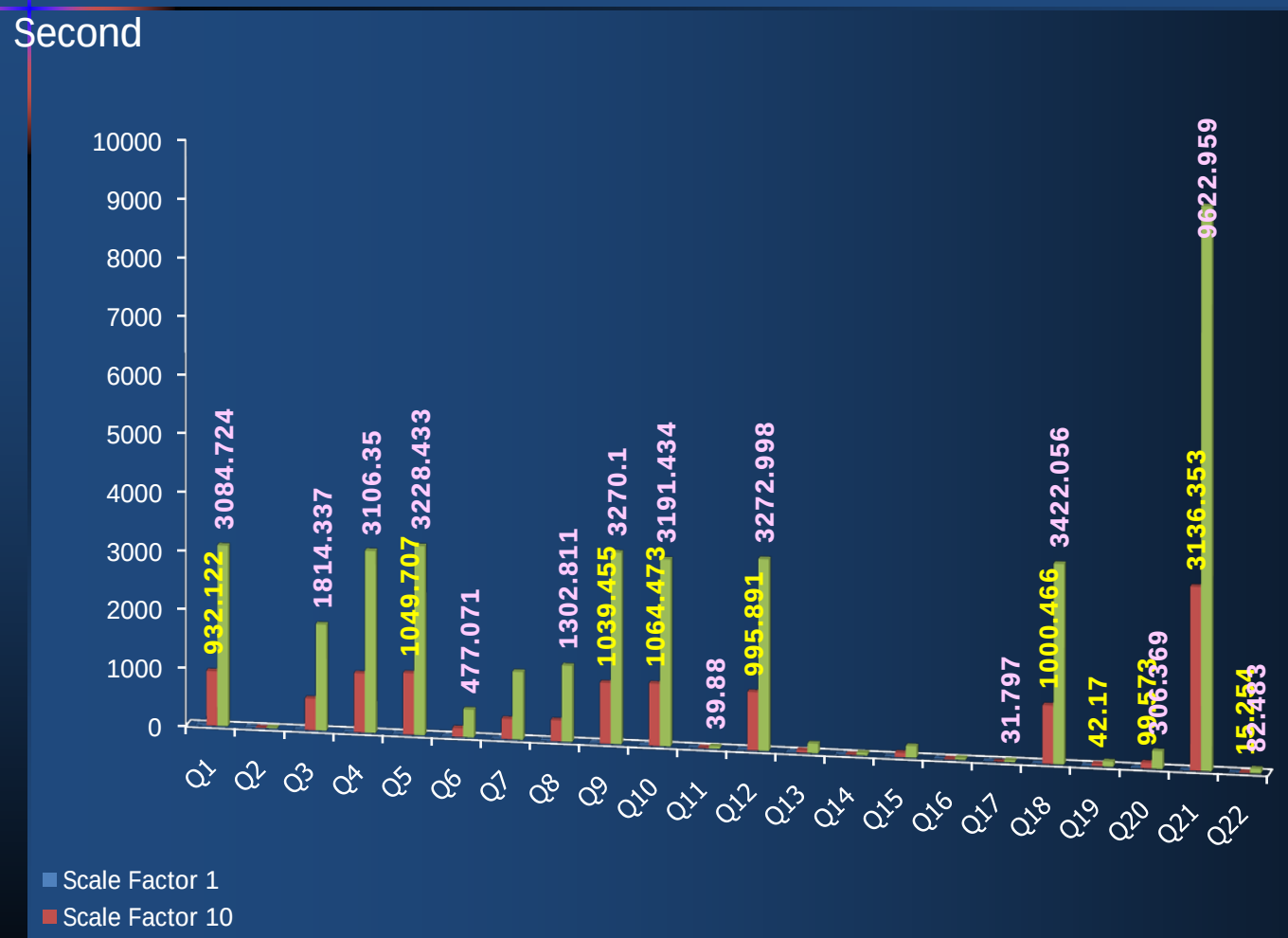
- 64 bit machine (proline)
- Intel core 2 Quad CPU @2.66Ghz each
- 4GB of RAM
- 500GB of Hard disk

Software:

- Microsoft SQL Server 2008, 64 bit (Enterprise Edition)
- Microsoft Windows Server 2008, 64 bit (Enterprise Edition)
- Microsoft SQL Server Profiler imbedded in SQL Server 2008
- Microsoft SQL Server Tuning Advisor imbedded in SQL Server 2008
- Benchmark Factory for databases version 6
- Spotlight for SQL Server Enterprise edition

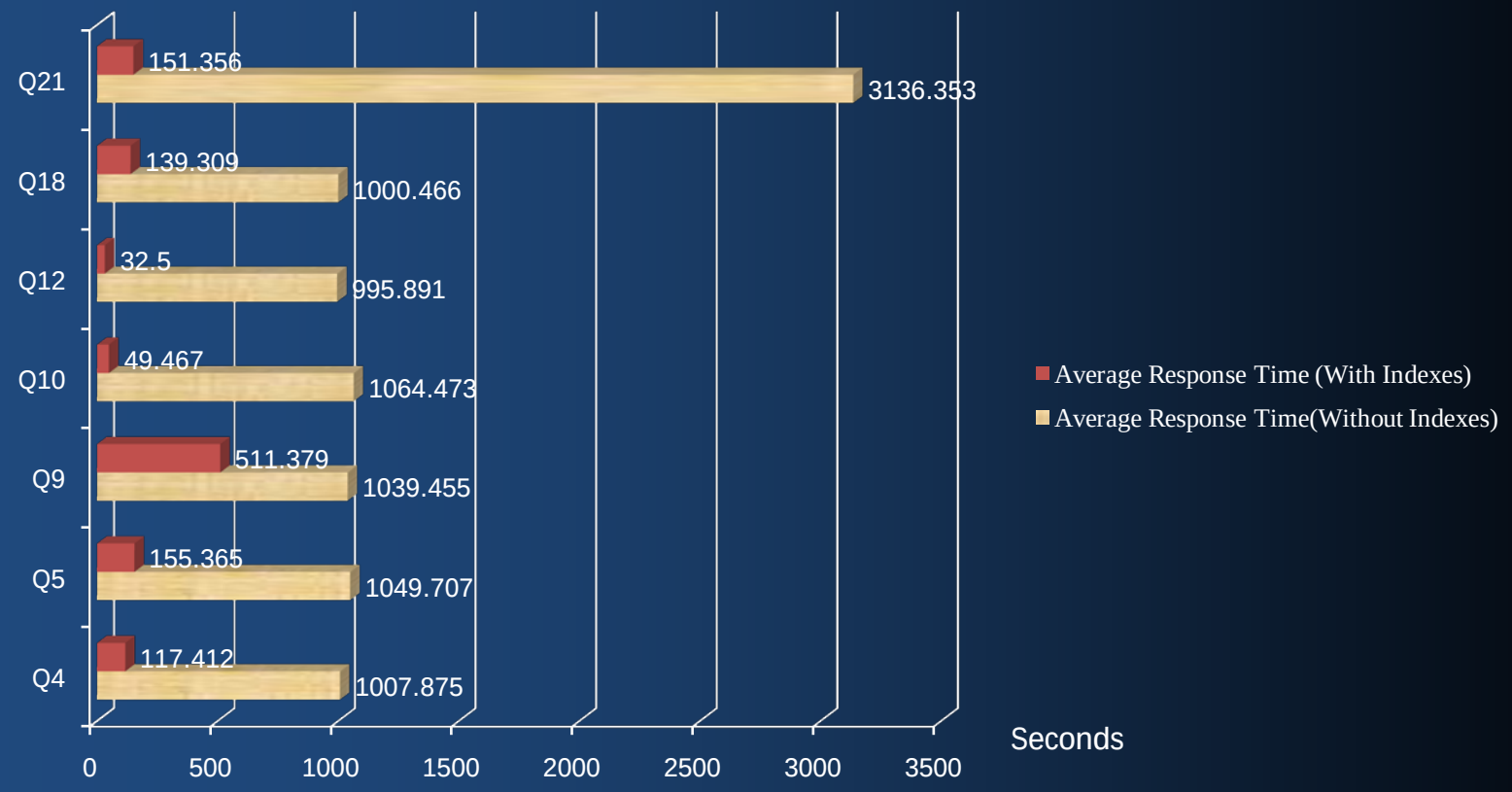
Results

With default configurations(Base line results)

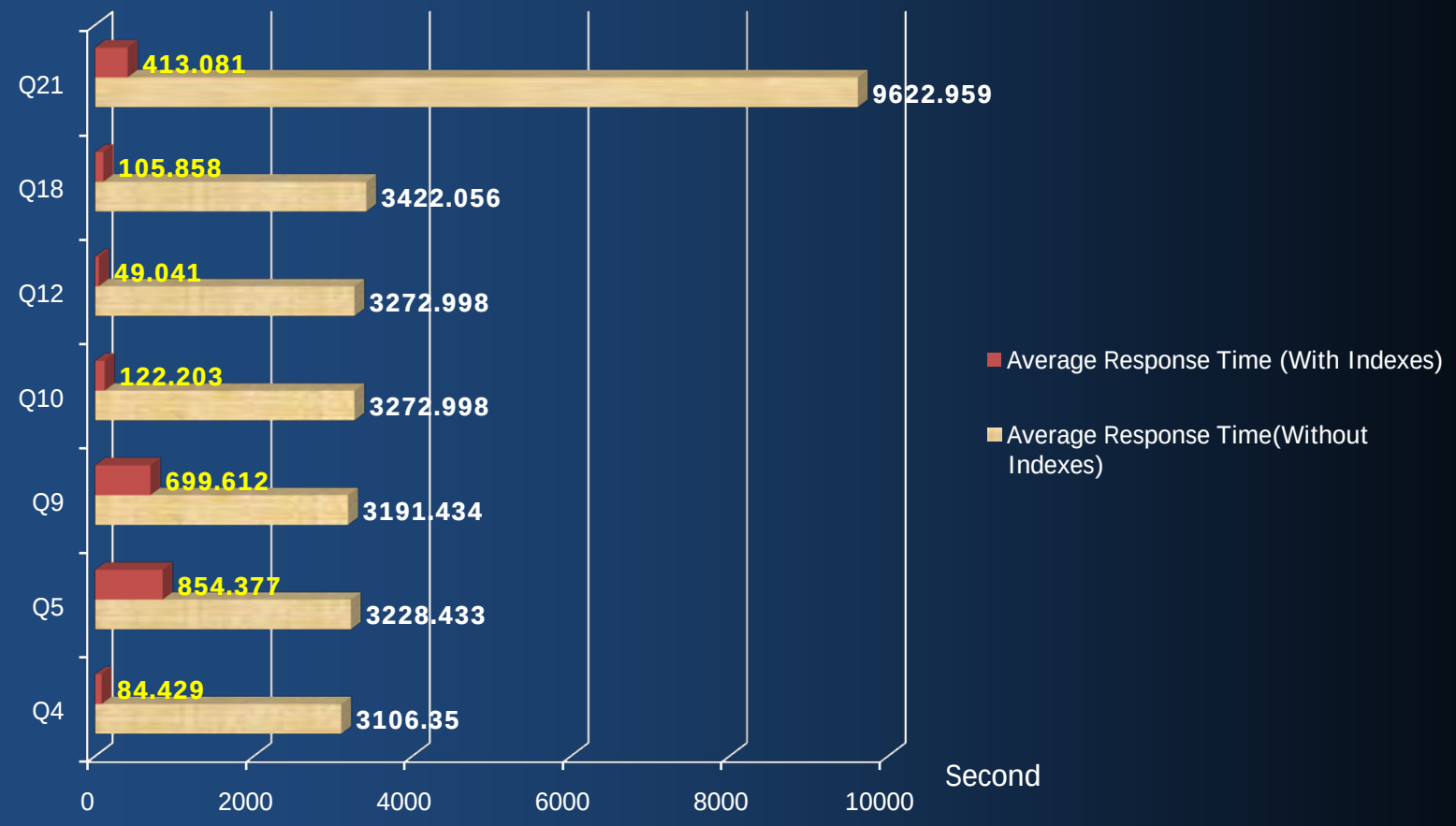


With Indexes suggested by the tuning advisor tool

Scale Factor = 10 / 10 GB



Scale factor = 30 /30GB



Suggested indexes analysis

Some concepts before embarking on to the actual analysis

- Indexes : used to minimize the numbers of disk read/sec so as to speed up data retrieval (They are designed by following B-tree algorithm for most databases including SQL Server)
- Clustered index: Point directly to the column on which is created and order the data in that column in a given manner. (a primary key is a good candidate)
- Non-clustered index : Point to the column(s) on which is created through the primary key (clustered index), good candidates are, foreign keys, column involves in join, Group by, Having)
- Covering Indexes: Are non clustered indexes in which extra columns have been included.

  Advantage: They are not considered being part of the index by the database engine as such, can contain data type not allowed for columns to be included in an index. They cover all the columns involved in query, hence the actual need not to be touched (good for performance)

Suggested indexes analysis cont…

Query4

```
select
        o_orderpriority,
        count(*) as order_count
from orders
where
        o_orderdate >= date '[DATE]'
        and o_orderdate < date '[DATE]' + interval '3' month
        and exists (
                select *
                from
                        lineitem
                where
                        l_orderkey = o_orderkey
                        and l_commitdate < l_receiptdate
                )
group by
    o_orderpriority
order by
    o_orderpriority;
```

Query 12

```
select
        l_shipmode,
        sum(case
                when o_orderpriority ='1-URGENT'
                        or o_orderpriority ='2-HIGH'
                then 1
                else 0
        end) as high_line_count,
        sum(case
                when o_orderpriority <> '1-URGENT'
                and o_orderpriority <> '2-HIGH'
                then 1
                else 0
        end) as low_line_count
from
        orders,
        lineitem
where
        o_orderkey = l_orderkey
        and l_shipmode in ('[SHIPMODE1]', '[SHIPMODE2]')
        and l_commitdate < l_receiptdate
        and l_shipdate < l_commitdate
        and l_receiptdate >= date '[DATE]'
        and l_receiptdate < date '[DATE]' + interval '1' year
group by
        l_shipmode
order by
        l_shipmode;
```

Suggested indexes analysis cont…

GENERATED INDEXES:

1.
```
USE [TPCH_10GB1]
GO
CREATE NONCLUSTERED INDEX [_dta_index_H_Lineitem_11_2137058649__K1_K12_K13] ON
        [dbo].[H_Lineitem]
(
        [l_orderkey] ASC,
        [l_commitdate] ASC,
        [l_receiptdate] ASC
)
ON [PRIMARY]
GO
```

2.
```
USE [TPCH_10GB1]
GO
CREATE NONCLUSTERED INDEX [_dta_index_H_Order_11_21575115__K1_K5_6] ON
[dbo].[H_Order]
(
        [o_orderkey] ASC,
        [o_orderdate] ASC
)
INCLUDE ( [o_orderpriority])
ON [PRIMARY]
GO
```

Indexes suggestion analysis cont…

3.
```
USE [TPCH_10GB1]
GO
CREATE NONCLUSTERED INDEX [_dta_index_H_Lineitem_11_2137058649__K15_K13_K11_K1]
ON [dbo].[H_Lineitem]
(
        [l_shipmode] ASC,
        [l_receiptdate] ASC,
        [l_shipdate] ASC,
        [l_orderkey] ASC
) ON [PRIMARY]
GO
```

4.
```
USE [TPCH_10GB1]
GO
CREATE NONCLUSTERED INDEX
[_dta_index_H_Lineitem_11_2137058649__K15_K13_K11_K12] ON [dbo].[H_Lineitem]
(
        [l_shipmode] ASC,
        [l_receiptdate] ASC,
        [l_shipdate] ASC,
        [l_commitdate] ASC
) ON [PRIMARY]
  GO
```

Findings (for indexes)

- The suggested indexes included foreign key columns, columns involved in joins, order and Group by.

- The Example of QUERY 4 AND QUERY 12, almost all the columns involved in these two queries are present in the indexes. Hence the queried data can be located directly from the index table instead of actual tables that contain million of records (hence, limiting the number of read per second from the disk).

- This explains to some extent the dramatic decrease of the time it takes to retrieve the data from the two databases(10GB and 30GB)

Optimization with parallel query processing

"Some concepts before embarking on to the actual process"

The main three ways in which query processing parallelism is implemented into databases are:

- Inter-query parallelism: Multi-users querying the database concurrently (OLTP)
- Inter-operator parallelism: one operator processes the data and passes it on to the next operator and so forth (pipeline parallelism)
- Intra-operator parallelism: The dataset is divided into chunks and an instance of a single operator is applied to each chunk.

  Microsoft SQL Server 2008 supports Intra-operator parallelism.

Optimization With parallel query processing cont…

Microsoft SQL Server configuration setting of interest:

- Max degree of parallelism : Determine the number of CPUs dedicated to SQL Server processes.

  MAXDOP=0 means all the CPUs are available for SQL Server processes, MAXDOP=1, Only one CPU available hence no parallelism.

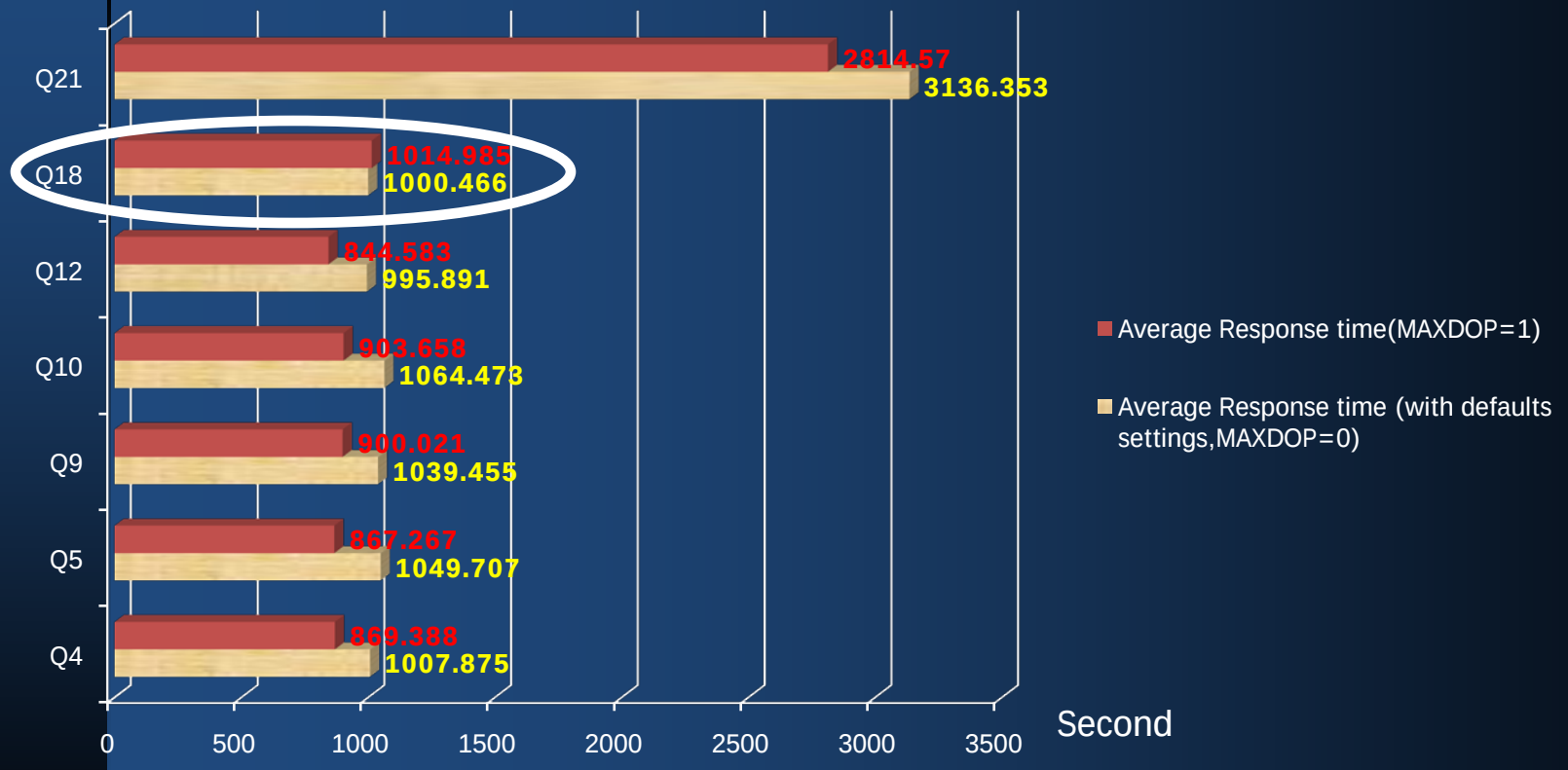
- Cost threshold for parallelism: The amount of time in second above which the query optimizer will opt for a parallel execution plan instead of a serial. The value of 5 seconds is the default one.

- The Max worker threads: Specifies the number of threads made available to SQL Server processes. A defaults value of 0 automatically configure at start up the number of thread for a 64 bit machine with <= 4 processor to 512. (The number of threads can be more than the number of processors)

Results with Parallel Query Processing

Scale factor=10 /10 GB

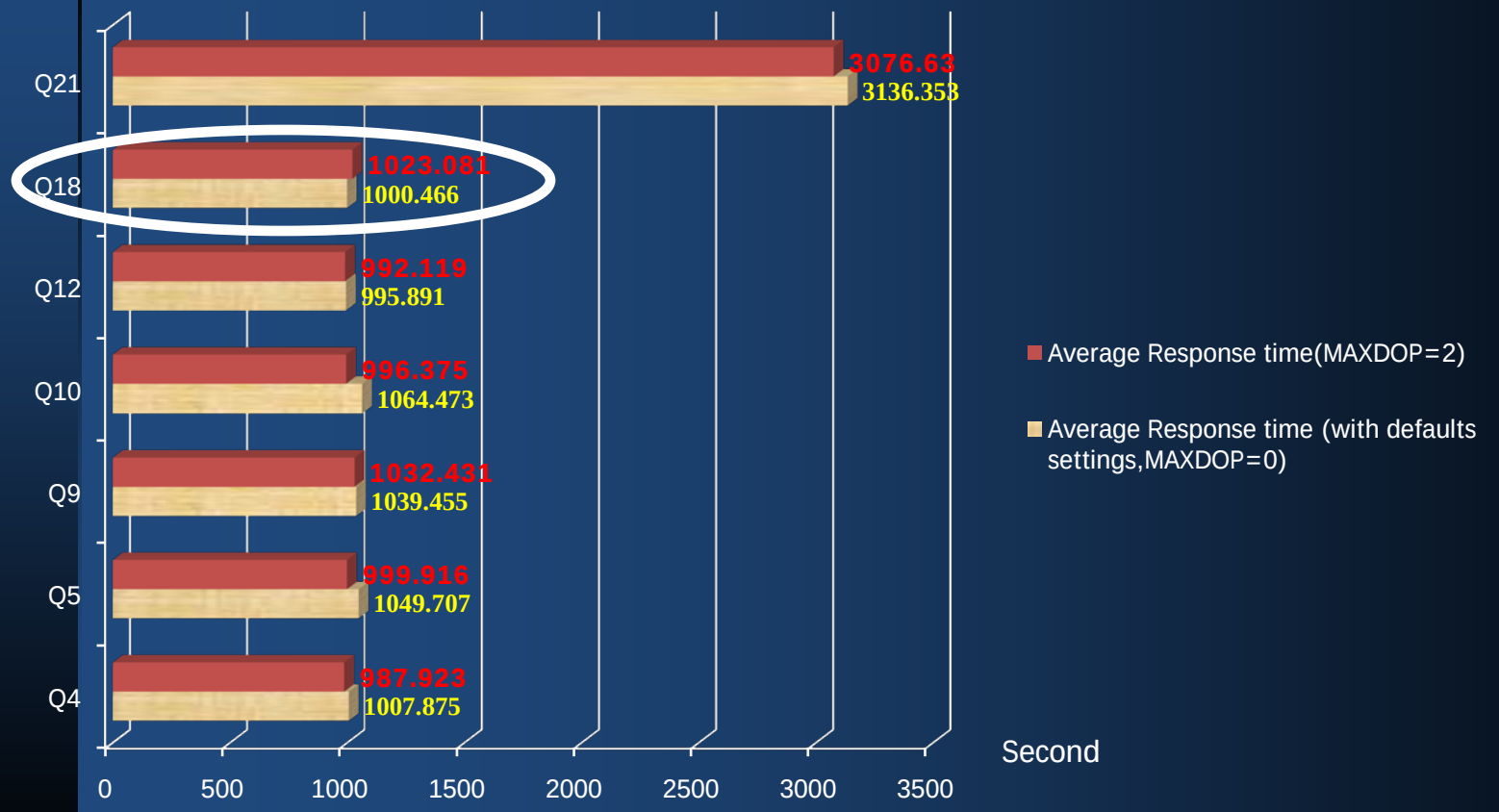
Max degree parallelism =1 vs Default settings (all for CPUs)



Results with parallel Query processing cont…

Scale factor=10 /10 GB

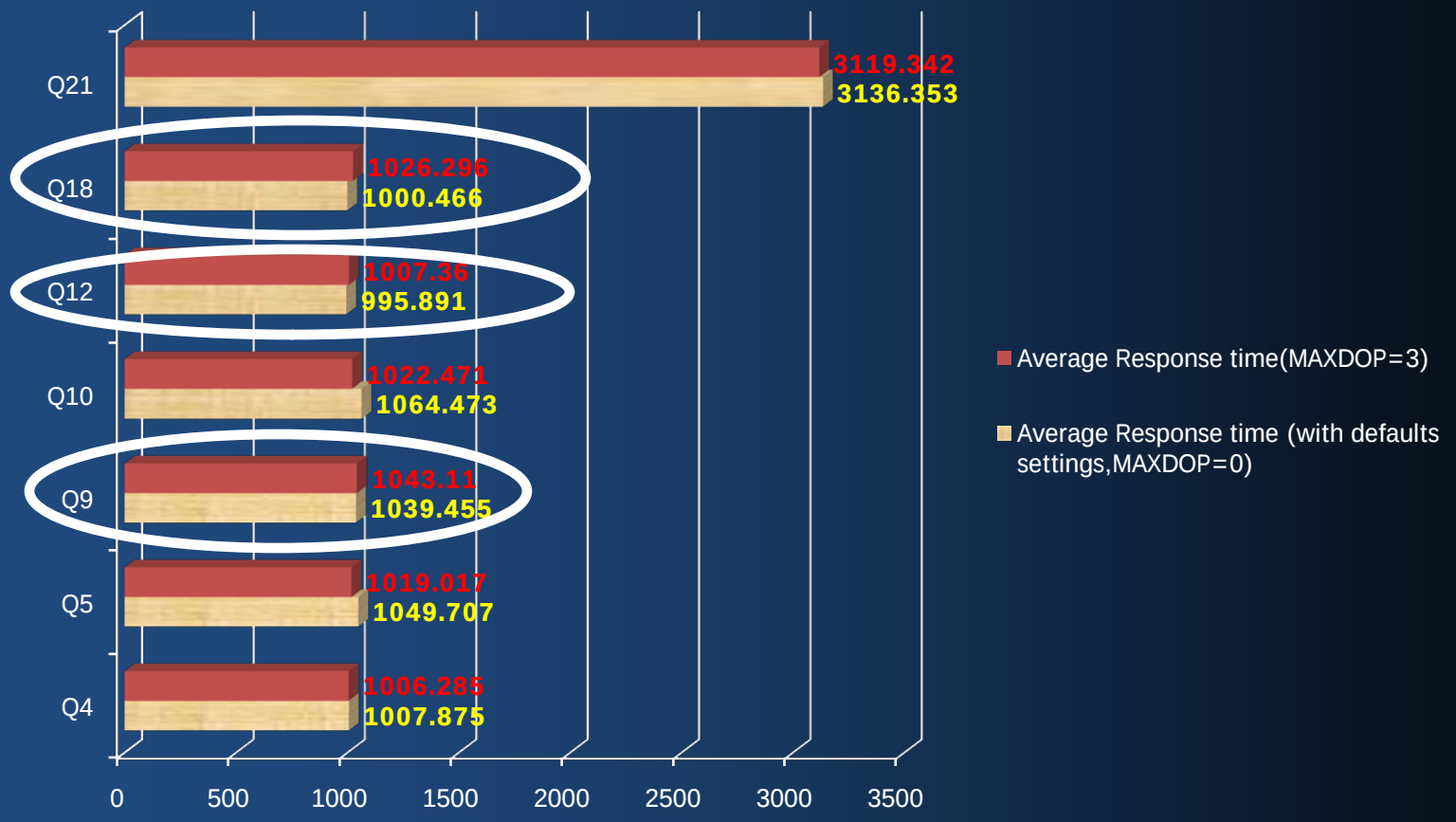
Max degree parallelism =2 vs Default settings (all 4 CPUs)



Results with parallel Query processing cont…

Scale factor=10 /10 GB

Max degree parallelism =3 vs Default settings (all 4 CPUs)



Findings (Parallel processing)

- Queries run sequentially execute faster than queries run in parallel.

- Executing queries with four CPUs takes longer than executing them with one, two, or three CPUs

- Where the execution of queries with four CPUs is faster compared to one, two, and three CPUs

The difference in execution time is just slight.

Quotes from SQL Server 2008 professionals (Blog)!!

“a problem that I had seen before, where according to the Microsoft engineer, "Yes, it's a bug - whenever a parallel query plan is generated”

“you should address the issue and validate / vote / comment so that Microsoft will have more motivation to treat these bugs seriously”

(http://sqlblog.com/blogs/aaron_bertrand/archive/2009/03/21/six-reasons-you-should-be-nervous-about-parallelism.aspx)

What is Microsoft saying about it?

- SQL Server defines a wait type “CXPACKET” in sys.dm_os_wait_stats stored procedure

- CXPACKET which occurs when synchronizing the query processor exchange operators.
- Query 4 scale factor =10

With MAXDOP=0 (ALL CPUs working),

Waiting _tasks_count=5348

Wait_time_ms=4885645 ms

With MAXDOP=1

Waiting_tasks_count=0

Waiting_time_ms =0

Recommendations

- When used appropriately indexes appear to be beneficial in the performance optimization of decision support workloads such as the TPC-H

- The Tuning Advisor built into SQL Server suggests indexes where appropriate. It is a tool to consider when planning the creation of indexes.
- Query Parallel processing on SQL Server 2008 needs to be examine carefully before being implemented on a production environment since running queries sequentially can be better than running them in a parallel. (Intra-query parallelism)

Future work?

- The Design of benchmark tool kits that are readily available for use on any OS application platform.
- Experimentation of techniques such RAID and Cluster for optimizing decision support workload
- Comparison of parallel query processing between SQL Server 2008 and Other database products (open source or proprietary)