Enhancing EHR Performance for Better Business Outcomes

Business Problem

An EHR with high volume of data and users often complains about performance glitches in certain sections of an application. Different types of performance issues encountered from end customers are:

- Slowness in application on executing a particular business scenario
- Slowness of the application during peak hours
- Poor response time with the application when number of concurrent users increases

The Analysis

Performance Bottleneck

A bottleneck is a point in a system where the flow of data goes down or the data flow is reduced. There is not enough data handling capacity to handle the current amount of data. A bottleneck can occur in an application where there is excessive usage for internal server resources. The common performance bottleneck factors are;

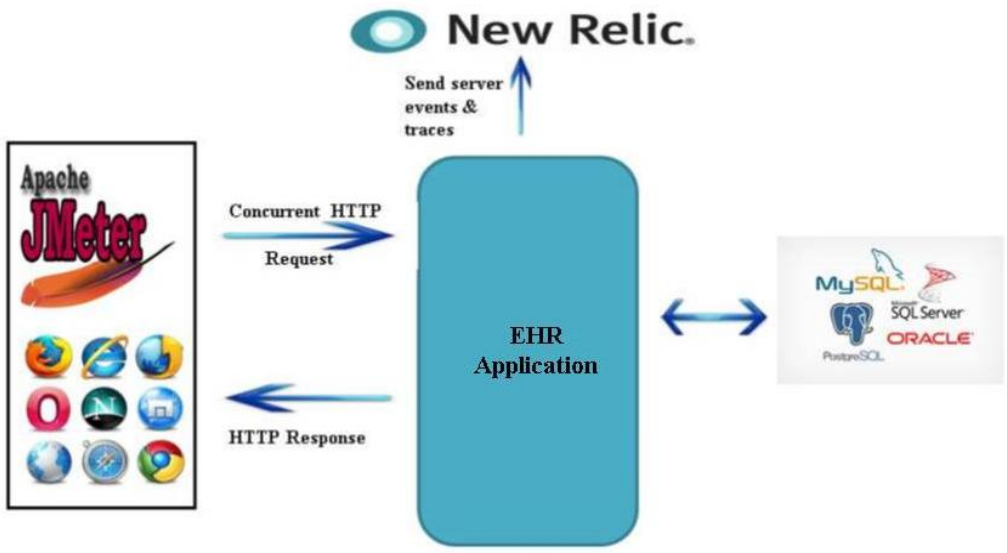
- CPU: CPUs can handle millions of instructions and calculations, but performance suffers when the numbers of operations exceeds the capacity. When CPU usage is greater than 75%, it will slow the entire system.
- Memory: Performance bottlenecks related to memory is due to **poorly designed software** (memory leaks) or other system flaws that leads to memory issues.
- I/O: I/O Wait is the percentage of time the processors are waiting on the disk to perform read or write operations.
- Network: The network bottleneck is a commonly blamed source of performance bottlenecks, but it is rarely found. It happens only when the size of **bandwidth is less**.
- Database: Database bottleneck can be caused due to **poor DB schema**, inefficient queries written within an application. DB that is not configured properly will also consume a lot of CPU and memory space.

Here, we consider Database bottleneck as a problem and provide solution for some of common problems.

Tools Used

In order to understand the application transactions that are slower and the reason for the performance degradation, two tools JMeter and NewRelic are used. Jmeter- an Open Source testing software, is used to simulate heavy load on servers to analyze the application behavior

under heavy load. New Relic- application performance tool is used to track the application response time, most time consuming transactions and database calls, which help us in identifying the problematic areas in the application to be fine-tuned. It also provides CPU Usage, Physical memory utilization, Disk I/O and Network I/O utilization.

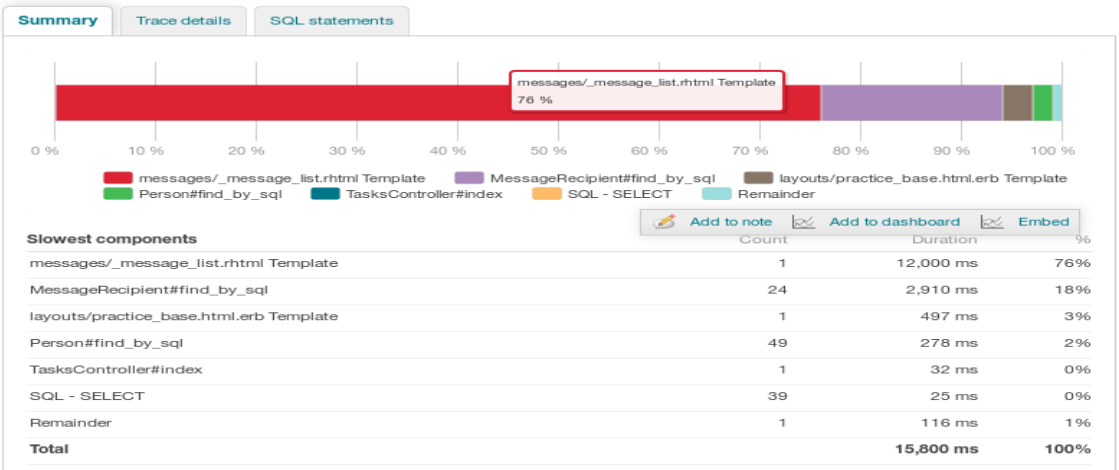


Identification of the performance issue

The following steps are used to analyse the performance issues using Jmeter and NewRelic.

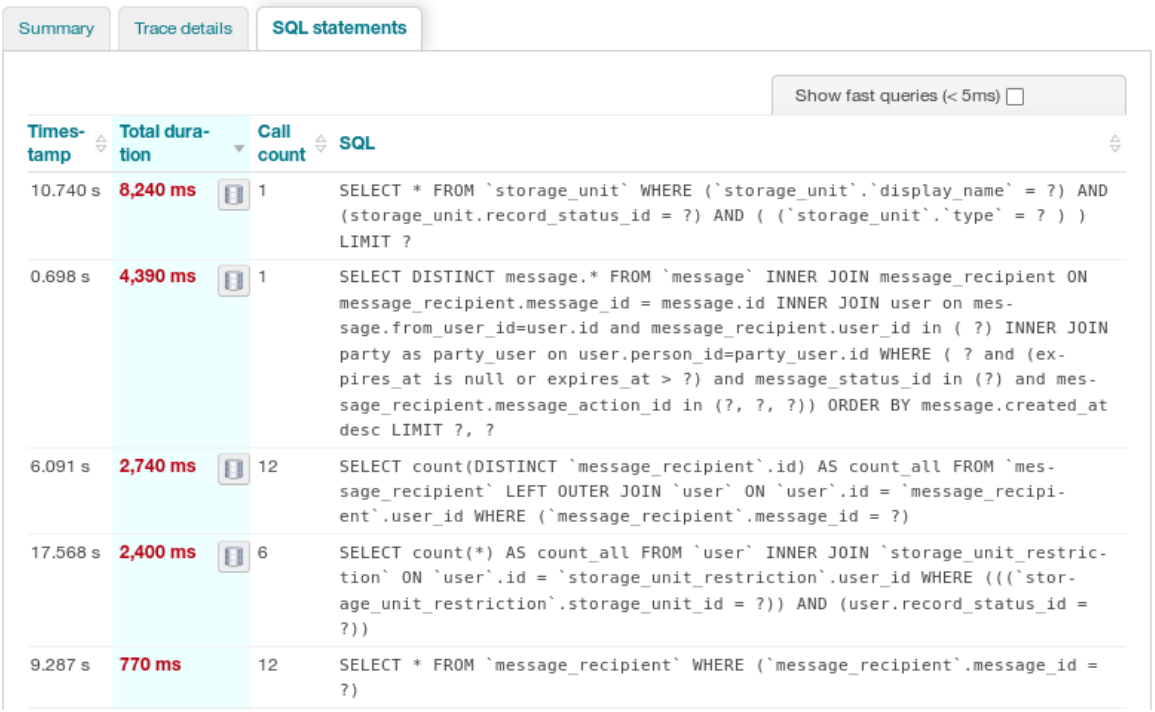
1. Setup JMeter in client machine
2. Record the clinical scenarios which needs to be fine-tuned for performance
3. Setup NewRelic in the EHR server and start the process.
4. Configure JMeter to simulate heavy load with the scenarios captured in (2)
5. Run the scenarios from JMeter
6. NewRelic will keep track of all the transaction details like slowest SQL queries, slowest transactions, application response time, time spent in database calls etc.

Sample graph obtained from New Relic with the list of slowest transactions and response time.



Slowest components	Count	Duration	%
messages/_message_list.rhtml Template	1	12,000 ms	76%
MessageRecipient#find_by_sql	24	2,910 ms	18%
layouts/practice_base.html.erb Template	1	497 ms	3%
Person#find_by_sql	49	278 ms	2%
TasksController#index	1	32 ms	0%
SQL - SELECT	39	25 ms	0%
Remainder	1	116 ms	1%
Total		**15,800 ms**	**100%**

Another graph from New Relic with details of SQL queries, number of times it is called and the duration.



Timestamp	Total duration	Call count	SQL
10.740 s	8,240 ms	1	SELECT * FROM `storage_unit` WHERE (`storage_unit`.`display_name` = ?) AND (storage_unit.record_status_id = ?) AND ((`storage_unit`.`type` = ?)) LIMIT ?
0.698 s	4,390 ms	1	SELECT DISTINCT message.* FROM `message` INNER JOIN message_recipient ON message_recipient.message_id = message.id INNER JOIN user on message.from_user_id=user.id and message_recipient.user_id in (?) INNER JOIN party as party_user on user.person_id=party_user.id WHERE (? and (expires_at is null or expires_at > ?) and message_status_id in (?) and message_recipient.message_action_id in (?, ?, ?)) ORDER BY message.created_at desc LIMIT ?, ?
6.091 s	2,740 ms	12	SELECT count(DISTINCT `message_recipient`.id) AS count_all FROM `message_recipient` LEFT OUTER JOIN `user` ON `user`.id = `message_recipient`.user_id WHERE (`message_recipient`.message_id = ?)
17.568 s	2,400 ms	6	SELECT count(*) AS count_all FROM `user` INNER JOIN `storage_unit_restriction` ON `user`.id = `storage_unit_restriction`.user_id WHERE (((`storage_unit_restriction`.storage_unit_id = ?)) AND (user.record_status_id = ?))
9.287 s	770 ms	12	SELECT * FROM `message_recipient` WHERE (`message_recipient`.message_id = ?)

The Solution

In this section, some of the common problems that may happen while accessing the database from the application and the proposed solutions are discussed.

Database Access Changes

- N+1 problem

  ***Problem*: Select N+1** is a data access anti-pattern where the database is accessed in a suboptimal way. While this would still work, it is highly inefficient.

  Example: Consider the following code

```
parties = Party.limit(10)
parties.each do |p|
        puts p.address
end
```

  This code looks fine at the first sight. But the problem lies within the total number of queries executed. The above code executes 1 (to find 10 parties) + 10 (one per each party to load the address) = 11 queries in total.

***Solution*:** By using **Eager Loading** mechanism, we had solved this problem.

Example:

```
clients = Party.includes(:address).limit(10)
```

- Project only required columns

***Problem*:** Some places of the code had queries which select all the columns of the table even if only a few columns were required in the subsequent steps.

Example:

```
select * from party;
```

***Solution*:** Removed this unnecessary projection of all table columns and had the query select only required columns. This resulted in a slight performance increase.

Example:

```
select id, name from party;
```

- Eliminate unnecessary processing

Problem: Get all the records from the table and reject the records based on some condition is a time consuming process. It really degrade the application performance.

Example:

```
usr = "select * from user";
loop u in usr
  if u.name == '%xxxxx%' and u.password == obj.encrypt("password")
            Collect the record;
   endif
end
return collected record;
```

Solution: To overcome this issue we had used only **join & where** clause and retrieved only the required records from the DB.

```
usr = "select * from user where name == '%xxxxx%';
loop u in usr
   if usr.password == obj.encrypt("password")
            Collect the record;
   endif
end
return collected record;
```

- Adding index to Queries:

We were also looking for queries that did not use any index and made that query to use an index to increase performance.

Database Configuration Changes

In addition, as explained earlier some of the bottlenecks can happen because of a poorly configured database. The following are some of the Database configurations that can be fine-tuned in MySQL.

max_allowed_packet	The max_allowed_packet setting determines the maximum size of a single packet. If we use very long BLOB columns or long strings, this setting must be large enough to handle them properly.
thread_cache_size	How many threads the server should cache for reuse. This variable can be increased to improve performance if you have a lot of new connections.
tmp_table_size	The maximum size of internal in-memory temporary tables. It helps to improve the performance if the SQL create temporary table.
innodb_buffer_pool_size	The size in bytes of the memory buffer InnoDB uses to cache data and indexes of its tables. The default value is 8MB. The larger you set this value; the less Disk I/O is needed to access data in tables.
max_connections	The maximum permitted number of simultaneous client connections. By default, this is 100.
innodb_log_file_size	This is the size of the commit log files. This is the main tuneable and you will want to set it quite high for performance reasons. By default it is 5MB
tmpdir	MySQL writes temporary tables to disk, populates them with intermediate results, and then query them again for the final result. It improves the performance by setting high volume memory partition location

Example:

Problem: Setting minimum value to max_allowed_packet may cause the performance issue if the SQL returns large number of records. The value should be a multiple of 1024.

Before fix:

```
max_allowed_packet=64M
innodb_log_file_size=5M
innodb_buffer_pool_size=128M
```

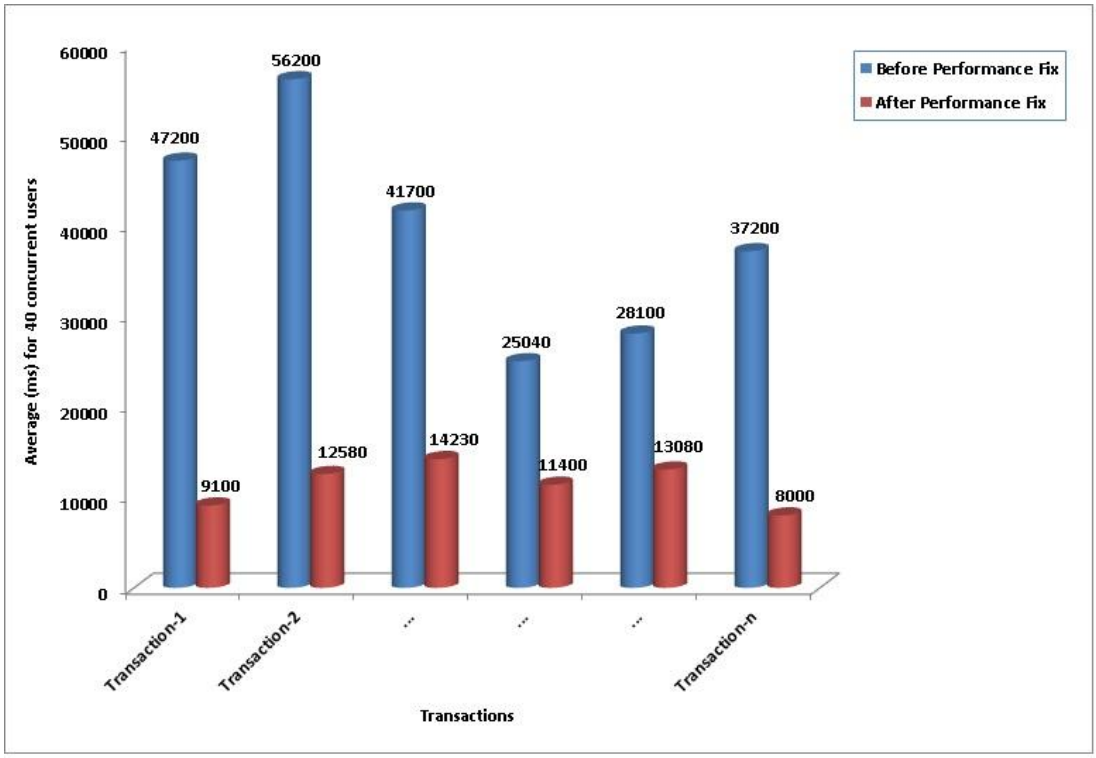
After Fix:

```
max_allowed_packet=10240M
innodb_log_file_size=128M
innodb_buffer_pool_size=2G
```

The Results

With help of Jmeter and New Relic, the slowest transactions are identified in the different sections of the EHR. After the analysis of the issues, data access pattern is modified and indexes are introduced. Database configuration is also fine-tuned for better performance.

Please refer the graph given below to shows the comparison report of the time taken for transactions before and after the performance fix. As you can see, the performance of the application had increased tremendously after the performance fine tuning.



About ViSolve

ViSolve, Inc., is a software services and consulting firm with expertise in Healthcare and Cloud. ViSolve is headquartered at San Jose, CA with best in class Development & Support center in Coimbatore, India. ViSolve is committed to provide better healthcare by providing vendor neutral IT services.

What we do?

- Cloud (VMWare, AWS, HP), DB (MongoDB / Hadoop), Migration, Enterprise Security, Performance, Deployment, Support and monitoring of customer environment with a focus on leading edge technologies
- Product development, build, QA and support of applications and kernel modules on VMware, HP-UX, Linux platforms
- 15+ years of experience in Open Source Solutions - Customization, Development and Support; leading contributor to the open source communities including Hadoop, OpenStack, OpenEMR, OSEHRA, Mongo DB, MySQL etc.

Why ViSolve?

- More than 15 years of strategic relationship with Enterprise Customers including Leading System Vendors, Financial Institutions ,Healthcare Organizations /Vendors
- Experience in successfully migrating over 160 healthcare customer environments to cloud with a focus on High Availability, Security, Disaster recovery, Performance and scalability
- As an extension of your R&D IT team - We Deliver Results of High Quality, On Time and Within Budget
- Expert Onsite Team & Low cost Offshore Team based in India that works in customer's preferred Time zone
- No 'one-size-fits-all' solution provider. Customization - Key Differentiator
- Key Advocate and Contributors of Open Source Communities including Customized Open Source Solutions with leading edge technology
- Access to a wide Pool of talented Engineers highly skilled in Leading Edge Technologies and extensive knowledge in Healthcare Domain
- We know our Customers, Partners and Technology inside-out, with clear perspectives on what is most fitting to solve our customer's challenges
- Avoid Vendor Lock in with Flexible Contractual Models. Competitive SLAs and Response Times, reduces internal cost & complexity and decreases Time to Market

To know more about how ViSolve can enhance your IT capabilities, get in touch with us by email at services@visolve.comor call us at +1 (408) 850 2243.