Goal driven performance optimization

When your goal is to optimize application performance it is very important to understand what goal do you really have. If you do not have a good understanding of the goal your performance optimization effort may well still bring its results but you may waste a lot of time before you reach same results as you would reach much sooner with focused approach.

The time is critical for many performance optimization tasks not only because of labor associated expenses but also because of the suffering - slow web site means your marketing budget is wasted, customer not completing purchases, users are leaving to competitors, all of this making the time truly critical matter.

So what can be the goal ? Generally I see there are 2 types of goals seen in practice. One is capacity goal this is when the system is generally overloaded so everything is slow, when you're just looking to see how you can get most out of your existing system, looking for consolidation or saving on infrastructure cost. If this is the goal you can perform general system performance evaluation and just fix the stuff which causes the most load on the system. MySQL Log analyzes with Mk-Log-Parser is a very good start for a ways to generally optimize MySQL load on the system.

Latency Goal is another breed. The system may not look loaded but some pages still may want to be loading much slower than you like. These goals are not system wise but they are much more specific to the different user interactions or even types of users. For example you may define goal also "Search pages have to have response time below 1 second in 95% cases and below 3 seconds in 99% cases". Note We're specific to the user interaction - people are used to Search taking longer time than other interactions for many applications, and also we speak about percentile response time rather than "all queries". It is surely good all search queries complete in one seconds but it is too not practical. The goal description may be more specific too - for example you may have different response time guidelines for pages which are requested for real humans vs search engine bots (which are often quite different in their access pattern) or you may define "large users" as users having more than 100.000 images uploaded and measure the response time for them specifically because this group has its own performance challenges.

Looking at Latency it is also much more practical to look from the top of the stack. If you look at MySQL log you may find some queries which are slow but it is hard to go back from them to what is really important for the user and so the business - the page response times. Furthermore. It is not enough in many cases to focus only on Server Side optimization - the Client Side Optimization is also quite important in particular for aggressive performance goals and fast back-end. This is why we added this service to Percona offerings.

If Server side or Client Side performance optimization is going to be more important for your application depends on the application performance a lot. The better your application is the more Client Side optimization you will need. For example if it takes you 30 seconds to generate the search results and 3 more seconds to load all style sheets images and render the page server side optimization is more important. If you have optimized things and now HTML takes 0.5 seconds to generates an extra 3 seconds become the main response time contributer which has the highest performance optimization potential.

But let us get back to the Server Side Optimization. Lets assume our performance goal applies to the HTML generation rather than full page load on the client. So meet our goal we should look at the pages which do not meet our goal, which is pages which take more than 1 second to generate in given example.

For goal driven performance optimization it is important there is enough instrumentation and production performance logging in place so you really can focus on hard data in your work. For small and medium size applications you can log all requests to MySQL table for larger ones you can log only small portion of them. I usually keep one table per day so it is easy to copy the data to a different box for data crunching and remove the old ones.

The log table should contain URL, IP and all the data you need to be able to repeat request if you need to. It may include cookie data, post data, logged in user information etc. But the real thing is number of times which are stored for request. wall clock time - is the real time it took to generate the page by server backend. CPU Time This is the CPU time needed to generate request (you can split it to user and system if you want) and when there come various wait times - mysql, memcache, sphinx, web services etc.

For web applications doing processing in a single thread the following simple formula applies wall_time=cpu_time+sum(wait_time)+lost_time The lost time is the time which was lost for some reason - some waits we did not profile or waits we do not have control of, for example when processing had to wait for CPU available to do processing. For multi-thread application it is a bit more complicated but you still can analyze critical path.

If you have such profiling in place all you have to do is to run the query to see what are contributing factors to the response time of the problematic pages:

SQL: mysql> SELECT count(*),avg(wtime),avg(utime/wtime) cpu_ratio, avg(mysql_time/wtime) mysql_ratio ,avg(sphinx_time/wtime) sphinx_ratio, avg((wtime-mysql_time-sphinx_time-utime)/wtime) lost_ratio FROM performance_log_081221 WHERE page_type='search' AND wtime>1;

count(*)	avg(wtime)	cpu_ratio	mysql_ratio	sphinx_ratio	lost_ratio
112376	6.0645327150223	0.11126040714778	0.17609498370795	0.54612972549309	0.16651488365119

1 row IN SET (2.29 sec)

Why looking only at such pages is important ? This is because if you look at all pages rather than problematic subset it may lead you away from your goal. For example it is very possible among all pages we would see CPU usage as the main factor because sphinx and MySQL respond from cache.

We however see for pages which have the problem it is Sphinx which accounts for most of the time.

Looking at the data such way we have two great benefits. First we really understand what is the bottleneck. Second we know what performance gain potential is. For example in this case we could spend a lot of time optimizing PHP code but because it takes only 10% of response time in average even speeding it up 10 times we would not get more than 10% response time reduction. At the same time if we find a way to speed up Sphinx we can reduce response time to its half.

Note in this case there is some 16% of response time which is not accounted for. Large portion probably comes from memcache accesses which are not instrumented for this application. In this case this portion is not the biggest part yet but if we'd speed up Sphinx and MySQL dramatically we would have to go and look into better instrumentation so we can look inside this black box.

Once we know it is Sphinx which causes the problem we have to go and find what queries exactly are causing it - this can be done by adding request ID as comment to Sphinx log so you can profile it carefully or you can add tracing functionality to the application. All the same. Once you found the queries causing the problem you see the ones which cause the most impact and focus on optimizing them.

There are multiple ways to optimize something, my checklist is usually get rid of it, cache it, tune it, get more hardware in this order. It is often it is possible to get rid of some queries, cache them, tune them so they are faster (often at the same time changing semantics a bit) and if nothing helps or can be done quickly we can buy more hardware, assuming application can use it.

Once you've performed optimizations you can repeat analyzes again to see if performance goals are met and where is the bottleneck this time.

As a side note I should mention looking at performance statistics for the day overall is often not enough. Application performs as good as it performs during its worst times so it is very good to plot some graph over time. Sometimes an hour base may be enough but for large scale application I'd recommend to looking down to 5 minutes or even 1 minute intervals and making sure there are no hiccups.

Check the stats from the application above for example:

SQL: mysql> SELECT date_format(logged,'%H') h,count(*),avg(wtime),avg(sphinx_time/wtime) sphinx_ratio FROM performance_log_081221 WHERE page_type='search' AND wtime>1 GROUP BY h;

h	count(*)	avg(wtime)	sphinx_ratio
00	5851	3.0608555987602	0.49142908242509
01	6639	2.9099249532198	0.48133478800683
02	5406	3.3770073273647	0.49140835595675
03	5397	2.9834221059666	0.53178056214228
04	4820	3.8182240369409	0.53530183347988
05	3720	13.025273085185	0.61126549080115
06	1606	60.624889697559	0.89123114911947
07	2699	38.821067012253	0.90885394709571
08	2419	45.388828675971	0.9226436892381
09	4810	6.330725168364	0.60329631087965
10	5445	3.8355732669953	0.53918653169648
11	5283	3.0498331333457	0.5512679788082
12	4147	2.9050685487542	0.52802563348716
13	2313	3.1297905412629	0.47887915792732
14	4155	2.9788750504185	0.53700871350403
15	4081	4.4940078389087	0.67605124513469
16	3720	3.1698921914062	0.54566719123393
17	4210	2.7616731525034	0.47537024159769
18	6735	2.639767089152	0.5204920072653
19	5581	2.6058266677645	0.42959908812738
20	4990	2.4441354725308	0.44270882435635
21	6305	2.6316682707403	0.5236776389174
22	6774	2.4394227009732	0.53342757714496
23	5270	2.3949674527604	0.51381316608346

24 rows IN SET (2.37 sec)

As you can see in this case during certain hours the average type of bad queries skyrockets and it becomes 90% or so driven by Sphinx. This tells us there is some irregular activity (cron jobs?) is happening and it affects Sphinx layer

significantly.

Such goal based from top to bottom approach is especially helpful for complex applications using mutliple components (like sphinx and MySQL) or multiple MySQL Servers because in these cases you often can't easily guess the component which needs attention. Though even for less complicated single MySQL server application there is often the question if it is MySQL server causing the problem or if application code needs to be optimized.

Originaly posted by Peter on mysqlperformanceblog