

Optimizing Blaise Servers for High-Concurrency Survey Environments

Karl Dinkelmann¹, Joris Bleus², Shane Empie¹, Judah Perillo¹, & Kelly Lieske¹

¹ University of Michigan - Survey Research Center

² Statistics Netherlands

1. Introduction

Optimizing Blaise servers for high-concurrency survey environments is a challenging endeavor, as maintaining efficient server environments is paramount, especially in an era of increasingly complex multimode surveys and rising demand for highly available applications. In this paper, we will discuss the history and scope of the problem as we experienced it at the University of Michigan. Our investigation has allowed us to gain insights into performance improvements, cost-effectiveness, and diminishing returns when scaling server environments. We will present up-to-date findings from load testing across various multi-server configurations. We will also provide initial findings from deploying a multi-server setup in production to inform best practices for future configurations. We will discuss workarounds and complementary strategies we have employed to mitigate server stress, such as concurrency monitoring, session management, and SQL replication. Finally, our goal is to provide some actionable guidance for organizations seeking to enhance server efficiency, balance costs, and meet the demands of high-concurrency survey environments.

2. Background

When using the Blaise 5 Server, one of the most significant risks to your survey is the underlying architecture of your server environment. Suppose your survey application or respondents accessing the survey exceed the limits of what your infrastructure can handle. In that case, catastrophe strikes: data may be lost, and survey participants are forced to wait long periods for pages to load, perhaps causing them to give up altogether. In general, it's not a place anyone wants to be.

We began seeing issues that appeared to be *concurrency* issues in 2021. This series of events has led us to look further into our server configurations. We first had a study in production, saving the data to SQLite on the server and began having issues with the write-ahead feature in SQLite, where it locked the database and the write-ahead logs would start to grow. This issue led us to move all our surveys to store data in SQL Server rather than SQLite. In the spring of 2023, we had two distinct events happen on a study, one where the Blaise Services began crashing unexpectedly, and the second was a concurrency event where the survey became virtually unresponsive to anyone in the survey during the event. Additionally, much of the data being collected at the time became corrupted or lost during the three hours at the height of the event. This issue was brought on unknowingly by the project that experienced the concurrency event, sending 2,600 emails and text invites (containing survey links) through the sample management system. This increased activity level contributed to the RAM and CPU usage, which hovered at sustained 90% and hit 100% several times during the event, which would have affected all the studies in production on the same Blaise server.

The Blaise services crashing resulted from offline Blaise synchronization hitting the Blaise server and a timeout, as our data was stored in a generic stream. The solution was to expose the primary key in a flat

table outside the generic stream data. However, the solution for the concurrency event is much more convoluted and becomes the origin of this paper.

Up to this point, our gold standard server setup configurations at the University of Michigan have been to have one front-end web server (holding the Data Entry, Resources, and Web roles) and one back-end management server behind the firewall (holding the Audit, CARI, CATI, Data, Publish, and Session roles) where our testing environments had 4 GB of RAM and our production environments had 16 GB of RAM on each server.

Figure 2.1: Previous Gold Standard Production Environment

Gold Standard Configuration:									
Server		Blaise Server Role						Notes	
		Audit	CATI	Data	Data Entry	Resource	Session		Web
1	16GB 4 cores	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	The front-end web server (respondent-facing); the login portal also sit on this server.
2	16GB 4 cores	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	The back-end Blaise server (behind the firewall); is where data resides and is secure.

The morning after the event, we upgraded the servers to 32 GB with 8 cores. CBS tests versions of Blaise and expects to be able to support 500-600 concurrent web respondents to access their test web survey during their load testing. We have been analyzing the IIS logs and the Blaise Write Interceptor logging data (shown below in Figure 2.2). These numbers do not include other surveys actively on the same server (four other production surveys are on the same server). Between 4:30 and 4:50 PM, we attempted to access the survey but could not get in, so we gave up after waiting ten minutes. Based on these numbers, we found that our previous gold standard could only support between 32 and 42 concurrent respondents – very different from the CBS guidelines. However, we do not know how many of the 2,600 invited respondents attempted to access the survey simultaneously; these numbers only reflect the number of respondents in the survey that sparked the event. The number of respondents who hit the survey was much higher, but as the event occurred, they could not access it. Our numbers only reflect those who were in the survey during the event.

Figure 2.2: concurrent users during the event, excluding cases under 1 minute

Total	29	37	33	37	34	35	30	30	16	15	29	33	33	31	28	30	33	37	38	38	40	40	44	39	36	36	31	32	30	23	24	23	20	19	16	13	14	10	7	
:	2	2	3	3	3	3	3	3	4	4	4	4	4	4	5	5	5	5	5	5	5	6	6	6	6	6	6	7	7	7	7	7	7	8	8	8	8	8	8	9
:	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0	
:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Time	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	
	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M

This event sparked a wave of work to test several configurations to develop a new set of gold standard configurations. Nevertheless, we can confirm that larger and complex surveys require significant Blaise server resources.

3. Problem

While we know we have experienced one concurrency event, we suspect we have experienced others over time; however, it's hard to tell without proper real-time monitoring and logging. Monitoring Blaise in real time is difficult to determine if and when these events occur. We discuss some of the ways we have attempted to bridge the gaps in monitoring in section 6 "Workarounds, lessons learned, and other factors to consider". First, we should look at the hardware recommendations for Blaise:

Development machines (to create questionnaires):

Minimum: Intel i3 or similar processor, 4GB of RAM, SSD

Recommended: Intel i5 or similar processor, 8GB of RAM, SSD (256GB or more)

Server (production environment):

Minimum: 2 core processor at 2.6GHz, 4 GB memory, SSD

Recommended: 2 or 4 core processor at 3.2GHz, 8 GB memory, SSD (256GB or more)

(Blaise Information Architecture Hardware Requirements, 2024)

Having experienced what happens when one hits a concurrency event, we set out to shore up our infrastructure to be as robust as possible to ensure this never happens again. As part of this work, we wanted to determine how to test how many concurrent respondents a given environment handles. Additionally, we tried to figure out ways to track concurrency issues and how to spot them. One of the best ways to deal with concurrency issues is to proactively prevent them from happening.

All surveys are not created equally and each survey on a given Blaise server contributes to the problem. Consider that one survey can support 200 participants in a given server configuration and another could support 10,000 in the same server environment. How does one monitor survey participants in both surveys, given the differences in how the survey design affects the survey performance from a concurrency perspective?

It is important to note that, regardless of the work done to ensure your survey infrastructure is set up the best way possible, there will always be ways to improve or tweak aspects to optimise the load one can handle. Therefore, we will delve deeper into the elements we have found along the way. However, we hope this inspires others to begin sharing their experiences, configurations, and questions to ensure that, as a collective of Blaise users, we can all benefit from a collective shared experience and can ensure we can optimize our environments for a better experience for everyone involved.

As discussed in the next section, the possible configurations are endless; although this is useful, it can also become hindering. Finally, we propose a new set of standard configuration settings applicable to all Blaise users. This work also helps CBS update the minimum server settings documentation for its end users.

4. Consideration

These include factors such as hardware resource constraints and the complexity of survey instruments. Key considerations for solutions include cost (licensing, hardware, maintenance), processing power, and the nature of server activities. We list several factors to consider below, and those include (but are not limited to):

- Price of Blaise Server licensing
- Virtual servers vs physical servers
- Price of hardware.
 - Amount of RAM
 - Amount of cores, processing power/speed
 - Blaise application is not multi-threaded (but it is unclear if this is causing an issue).
 - Blaise web runtimes do support multi-threading. This could mean more cores and fewer servers (but requires more testing).
 - Networking gear and connections
- The cost of resources to maintain the servers
 - Do the right people have the right level of access to the servers?
 - How many versions of Blaise need support?
 - Testing upgrades to ensure outcomes meet expectations
 - Windows Server upgrades
 - SQL Server upgrades (or database choice)
 - Blaise Server upgrades
- Effect of additional security. For example, Cloudflare (www.cloudflare.com), CrowdStrike (www.crowdstrike.com), Firewalls, and other security features.
- Power configurations of servers.
- How many studies are on the server? How many concurrent users? Other things happening on the server (such as write interceptor hits during interviewer syncs), though the back-end is fine, and the load mainly affects front-end activity.
- Not all surveys are created equally (complexity, load). One survey can support 1000 concurrent users, and another resource-intensive survey could support 140 concurrent users.
- Moving large or resource-intensive surveys onto their own Blaise server set. This can help both the resource-intensive and the surveys remaining on the previous server set. However, this does require an additional Blaise license.

5. Load Testing

The University of Michigan's experience with load testing Blaise instruments was minimal before our initial concurrency event. However, it became apparent that something needed to change and the next natural step was to gather empirical evidence from formal load testing with different server configurations. As part of our efforts to gather more information for this paper, we reached out to CBS and asked for the survey they use to load test versions of Blaise so we could begin to develop a baseline for our different environments.

The University of Michigan's current work is on Blaise versions 5.14.4 and 5.14.9. However, CBS load tests all versions of Blaise. Below, we outline the steps to execute load testing on Blaise surveys.

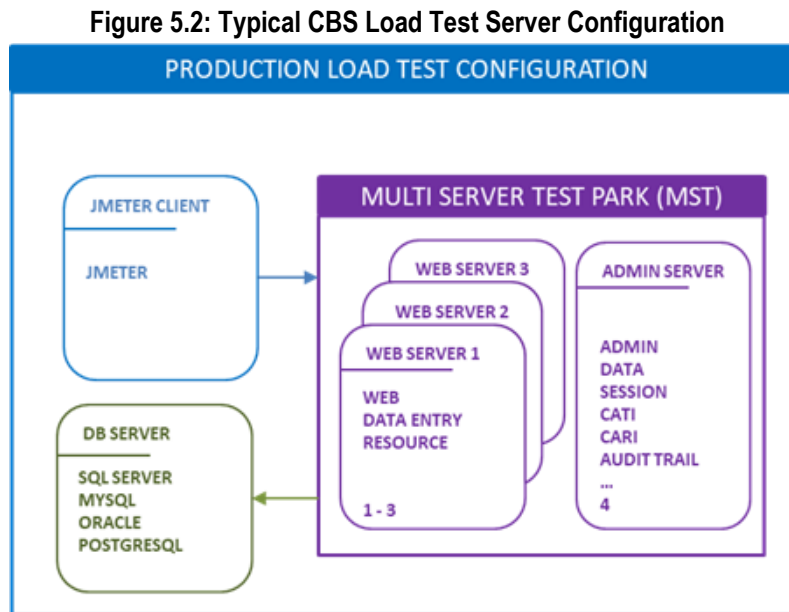
5.1. Applications needed

The following are what is needed to run load tests.

- **Required:**
 - **JMeter** (<https://jmeter.apache.org/>) - to run the tests (Free)
 - **Java** - needed for JMeter (Free)
 - **BlazeMeter** (<https://www.blazemeter.com/>) - to record a web survey case (Free)
- **Advanced users (optional and additional cost):**
 - **Ranorex** (<https://www.ranorex.com/>): CBS uses Ranorex for test automation. However, it can also be used for UI testing. (Licence cost)
 - **Azure Pipelines** (<https://azure.microsoft.com/en-us/products/devops/pipelines>) could be automated via the pipeline. (Licence cost)

5.2. Setup:

Setting up the Blaise Load test environment. CBS uses Ranorex as our test tool shell to set up the Load test environment. Within Ranorex, different steps are executed using Ranorex features, command-line scripts, C# code, etc. In general, this is what our load test server configuration looks like:



The load test is automated at CBS, meaning it's part of the test cycle. With every (test) build, all tests are automatically scheduled for a nightly run on our dedicated test environment. Below, we will go into the specifics about the CBS test process in general and attempt to highlight the relevant steps for the load test.

Next to the dedicated Blaise load test servers, there's a database (DB) server and a JMeter client server. The DB server is configured with a SQL Server, MySQL, PostgreSQL, and Oracle database. This

machine is used for all database-related tests, including the load test. The JMeter client is on a test production machine from which the performance tests are run (amongst others). Four load tests are run, one for each database as described above.

These are the steps to set up the load test environment for one of these databases:

1. Based on the Blaise test version, the associated test sources are automatically cloned from GitHub to the JMeter client machine. These test sources contain the Ranorex Load test solution, the datamodel, manipula setup, JMeter sources, and config files.
2. The Blaise MSI is sent to all servers in the Multi Server Test Park (MST).
3. Blaise is then uninstalled in MST, removing all references (like registry entries).
4. Blaise is installed with an HTTPS configuration on every server. Three web servers have the roles WEB, DATAENTRY, and RESOURCE. The admin server has the other roles.
5. A server park is created, and three web servers are added to the admin server.
6. The Blaise load survey is built, bdix files are created for the specific database, and pre-filled data is generated. The bdi files for session (bsdi) and audit trail (badi) data are also created and set in the local server settings. In this process, an internal tool called 'DataInterFaceCreator' is used for making the bdix files. Dropping and creating tables is also executed with this tool.
7. The survey is then installed in the server park, verifying the correct installation. Once the verification is done, the load test starts.

See the appendices below for more information on the listed topics.

- Appendix A: Blaise install Command-line
- Appendix B: Load test setup at CBS
- Appendix C: Ensuring Unique Key Value per Thread in an Infinite Loop in JMeter

5.2. Preliminary results:

Figure 5.2.1.1 16GB RAM vs 32GB RAM

Requests	Michigan	Executions					
Label	#Samples	FAIL	Error %	Average	Min	Max	
Total	120540	0	0.00%	2238.50	1	14681	
executeaction	105540	0	0.00%	2347.39	36	14681	
fetch_lookup_results	600	0	0.00%	2372.94	16	8607	
gezoexp	1200	0	0.00%	3427.71	126	14603	
key_page	1200	0	0.00%	2971.65	89	6433	
receipt_page	600	0	0.00%	4891.07	2306	13194	
select_external_record	600	0	0.00%	2219.44	44	8636	
start_interview	1200	0	0.00%	2825.38	71	6660	

Requests	Blaise	Executions					
Label	#Samples	FAIL	Error %	Average	Min	Max	
Total	146609	0	0.00%	77.75	0	1167	
executeaction	130409	0	0.00%	80.69	22	1167	
fetch_lookup_results	1200	0	0.00%	23.59	8	272	
gezoexp	1200	0	0.00%	80.47	43	595	
key_page	1200	0	0.00%	159.96	56	772	
receipt_page	600	0	0.00%	233.69	64	855	
select_external_record	1200	0	0.00%	49.21	23	304	
start_interview	1200	0	0.00%	155.87	48	768	

We ran one SQL Server load test with build 5.14.6.3686 on our newly configured servers with 32GB RAM and compared the results with our 16GB RAM on all servers with a 5.14.6.3686 baseline. As you can see, the performance is slightly better but not much: 7,41% faster for the TOTAL label (all requests average). The performance increase could be more substantial if we hit more concurrency; unfortunately, we don't have these figures. This also concerns one run for both the baseline and the new test. It may take several runs to determine the difference between them.

Figure 5.2.1.2 16GB RAM vs 32GB RAM

00:00.052	Info	Data	Current variable values: \$numberofusers = '1800', \$BaselineReleaseNr = '5.14.6', \$db = 'SQLServer'
00:00.132	Success	Validate JMeter CSV file	executeaction baseline:80 and executeaction current:74 has an acceleration of -7,5% compared to baseline:80, an acceleration falls within the norm of 33% Baseline 16GB vs Current 32GB RAM 5.14.6.3686
00:00.198	Success	Validate JMeter CSV file	fetch_lookup_results baseline:27 and fetch_lookup_results current:23 has an acceleration of -14,81% compared to baseline:27, an acceleration falls within the norm of 33%
00:00.258	Success	Validate JMeter CSV file	gezoexp baseline:78 and gezoexp current:101 has a delay of 29,49% compared to baseline:78, a delay falls within the norm of 33%
00:00.322	Success	Validate JMeter CSV file	key_page baseline:136 and key_page current:123 has an acceleration of -9,56% compared to baseline:136, an acceleration falls within the norm of 33%
00:00.386	Success	Validate JMeter CSV file	receipt_page baseline:143 and receipt_page current:129 has an acceleration of -9,79% compared to baseline:143, an acceleration falls within the norm of 33%
00:00.448	Success	Validate JMeter CSV file	select_external_record baseline:55 and select_external_record current:51 has an acceleration of -7,27% compared to baseline:55, an acceleration falls within the norm of 33%
00:00.506	Success	Validate JMeter CSV file	start_interview baseline:142 and start_interview current:129 has an acceleration of -9,15% compared to baseline:142, an acceleration falls within the norm of 33%
00:00.575	Success	Validate JMeter CSV file	TOTAL baseline:81 and TOTAL current:75 has an acceleration of -7,41% compared to baseline:81, an acceleration falls within the norm of 33%

5.2.2 Four CPU Cores vs Eight CPU Cores

We then ran a local load test (on one machine) with a basic Flight data model. One with four cores and 1 with eight cores. As you can see, performance increases when eight cores are used instead of four. Still, this is just a small test.

Figure 5.2.2.1 Flight w/4 Cores

Requests	Executions			Response Times (ms)					
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct
Total	19964	0	0.00%	2993.88	0	14897	2631.00	6375.00	7325.00
executeaction	6747	0	0.00%	3697.13	46	8159	3871.00	6420.00	6956.00
flight/	1505	0	0.00%	6697.42	8	14897	6890.00	11584.20	12760.00
flight/-0	1505	0	0.00%	3391.31	0	7878	3423.00	5937.20	6689.00
flight/-1	1505	0	0.00%	2.93	0	44	2.00	6.00	8.00
flight/-2	1505	0	0.00%	3.23	0	48	2.00	6.00	9.00
flight/-3	1505	0	0.00%	3296.36	1	7737	3342.00	5841.20	6446.00
flight/-4	1505	0	0.00%	3.13	0	72	2.00	6.00	9.00
flight/-5	1505	0	0.00%	3303.58	6	7738	3365.00	5847.00	6446.00
receiptpage	1232	0	0.00%	3755.05	34	7891	4002.00	6321.40	6950.00
start_interview	1450	0	0.00%	3495.66	28	7918	3647.50	6188.50	6637.00

Figure 5.2.2.2 Flight w/8 Cores

Requests	Executions			Response Times (ms)					
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct
Total	24085	0	0.00%	2234.66	0	10778	2847.50	4864.90	5571.00
executeaction	8211	0	0.00%	2800.85	38	5761	3092.00	4564.00	5280.00
flight/	1800	0	0.00%	4886.22	7	10778	5571.50	8402.70	9289.00
flight/-0	1800	0	0.00%	2454.67	0	5508	2742.50	4248.00	4909.00
flight/-1	1800	0	0.00%	1.62	0	24	1.00	3.00	5.00
flight/-2	1800	0	0.00%	1.75	0	37	1.00	3.00	5.00
flight/-3	1800	0	0.00%	2423.84	1	5396	2722.00	4208.90	4824.00
flight/-4	1800	0	0.00%	1.79	0	24	1.00	3.00	5.00
flight/-5	1800	0	0.00%	2430.07	5	5408	2729.00	4213.80	4830.00
receiptpage	1515	0	0.00%	2798.28	32	5656	3154.00	4510.00	5219.00
start_interview	1759	0	0.00%	2629.13	27	5628	2932.00	4408.00	5013.00

Figure 5.2.2.3 Flight w/4 Core - 600 Users with Balanced Power

Requests	Executions			Response Times (ms)				
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct
Total	145345	0	0.00%	177.53	0	6678	71.00	228.00
executeaction	129145	0	0.00%	183.69	22	6678	72.00	226.00
fetch_lookup_results	1200	0	0.00%	89.40	7	1488	32.00	218.00
gezoexp	1200	0	0.00%	258.69	43	3466	68.00	698.50
key_page	1200	0	0.00%	344.68	60	3673	151.00	804.90
receipt_page	600	0	0.00%	586.86	84	3639	443.00	1089.70
select_external_record	1200	0	0.00%	95.78	23	1485	50.00	213.70
start_interview	1200	0	0.00%	325.03	52	3459	157.00	787.00

Figure 5.2.2.4 Flight w/8 Core - 600 Users with Balanced Power

Requests	Executions			Response Times (ms)				
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct
Total	146609	0	0.00%	77.75	0	1167	53.00	127.00
executeaction	130409	0	0.00%	80.69	22	1167	55.00	128.00
fetch_lookup_results	1200	0	0.00%	23.59	8	272	18.00	34.00
gezoexp	1200	0	0.00%	80.47	43	595	46.00	184.00
key_page	1200	0	0.00%	159.96	56	772	103.00	362.90
receipt_page	600	0	0.00%	233.69	64	855	184.00	455.90
select_external_record	1200	0	0.00%	49.21	23	304	39.00	82.00
start_interview	1200	0	0.00%	155.87	48	768	99.00	373.00

6. The New Gold Standard for the University of Michigan (as of print):

Based on our work so far, we have defined our new gold standard Blaise server configuration as shown below in Figure 6.1. However, we will reassess these standards annually, in the context of our project configuration and requirements, to determine if we need to make any adjustments.

Figure 6.1 The New Gold Standard

Environment	Servers				
	Management	Cores	Web	Cores	RAM
Old Gold Standard					
Testing	1	4	1	4	16 GB
Production	1	4	1	4	32 GB
New Gold Standard					
Testing	1	4	2	4	16 GB
Production	1	4	5	8	32 GB

7. Workarounds, lessons learned, and other factors to consider

One of the most significant aspects that we felt this paper needed to be written about is due to how much we have learned along the way, and we felt the Blaise user community could benefit significantly from the content. Therefore, we wanted to describe elements we captured along the journey. This list is not exhaustive; some items are likely specific to the University of Michigan network. However, the elements might be of assistance to other organizational infrastructures. Workarounds and things that are helping us:

- Monitoring concurrency is difficult; we have attempted to do this by adding custom logging to the Blaise Write Interceptor attached to our most demanding surveys. This allows us to parse out concurrency numbers to understand (after the fact) how many respondents are in a study at any given minute. Our current process is to run the script every morning for the previous twenty-four-hour period to see if we have anything to be concerned about. Alternatively, one could do something similar by parsing audit trail information. This can also be done retroactively; we have created a Python script to review historical Audit Trail data to see the peak number of respondents in a survey at any given point in a previous wave of data collection (or a time when we had not been actively thinking about concurrency issues).
- Our Information Technology (IT) team has been able to set up alerts in the past for us using a tool called "ipSentry" (<https://ipsentrydiscontinuedsa.z13.web.core.windows.net/>); although they have discontinued their product in 2024, their site lists other potential alternatives. However, as the University of Michigan began moving sites behind CloudFlare (<https://www.cloudflare.com>), our use of ipSentry decreased.
- We have begun adding a feature to our web survey login portals that queries the web survey for active sessions in the last 15 minutes in the Session Database; if the number is at or above a certain threshold, we present a message to the respondent letting them know the survey is currently full and they should come back later in the day and try again. The threshold is determined by our best estimate of the number of users the current server's configuration can handle for a given survey.
- We have had to start breaking survey invites up into more manageable packets. Instead of sending 5,000 invites to a survey at one time, they are broken into multiple sets of invites over a

ten-day period, where we send 500 invites a day. This is intended to spread out participants' responses, reducing the number of concurrent hits to the survey portal.

- We have moved all Blaise production data to a secondary server that uses SQL Server Replication, so the only entity natively touching Blaise data is Blaise. All humans access the replicated SQL data, removing the chance that any live data is not queried.
- Our server balancing is done via CloudFlare. However, one thing that is important to note is that when you load balance your servers, you must realise that Blaise is not stateless. This means the survey will break if you have a server jump during an active session. We have gotten around this by giving our DNS/URL aliases. For example, let's say we have a URL for a study, that is www.myBlaiseSurvey.com. This DNS would pass through CloudFlare, and CloudFlare would assign it one of the web servers being used in this multi-server environment. Once the server assignment is made, we use a DNS redirect to the URL alias on the server where the respondent lands. Let's assume they land on server number 3, the URL redirect takes this respondent to www.myBlaiseSurvey-3.com, so we can ensure this respondent stays on the same server during this given survey session.
- While we have not confirmed this as an issue, we have long thought of Blaise Lookups as having a problem if they use SQLite via web surveys, and as a result, we always attempt to make our web Blaise lookups so they use SQL Server instead. This can also be done in mixed-mode studies, where you can use the Mode variable to determine which lookup is used. For example, you have an offline thick-client version that points to the *.bdix file representing the SQLite version and an online thin-client version that points to the SQL Server version of the *.bdix file.
- IPerf - JMeter server monitoring agent
- We have yet to do Blaise syncing simultaneously while we are doing web load testing.
- Cloud Future? Increasing Cores may mitigate the need for more front-end servers.
- The University of Michigan has had to turn off the audit trail while testing. This was because a conversion script, ModifyJMeterTest.exe, needed to be updated to include the audit trail.
- Sometimes, Blaise services hold onto memory after running a test. Need to reset Blaise services before testing to get a more accurate reading.
- There is some manual configuration using the current batch files from Blaise to convert .jmx files to a setup for testing.
- Once load testing is built into your ecosystem, it should become something you do with every survey before entering production.

8. Wishlist

This work has brought forth some specific wishlist items that we think should be added to Blaise and these are as follows:

- Cross-survey monitoring dashboard on a given Blaise server – the Blaise Server Manager does not have insight into all the surveys installed on a given Blaise server; we need to drill into each survey individually. We feel we should be able to see all the surveys installed in real-time, which have active participants, the resources being used on the various servers that are part of the server cluster, etc.
- Custom server alert mechanism - creating custom alerts that send notifications to Slack and log information as JSON.

- Better error messaging that is more human-readable and accessible
- Better Data Viewer performance, as the Data Viewer is critical during testing in both case selection and outcome verification
- Basecamp area for load testing results and server configuration discussions.
 - Create a standard setup guide so users can get a baseline to compare with Statistic Netherlands and other user baselines.
 - And a user group that would actively participate in the discussions, best practices, workarounds.

9. Future

Many of our issues could be addressed if we moved to the cloud. Even though this poses additional challenges and resource needs, it remains on our long-term Blaise roadmap to investigate. We would like to explore what would happen if we split off specific surveys to their dedicated front end servers, where, for example one resource intense survey would be on its series of servers (like web servers 4-10) and all the other surveys on the same Blaise server were on a different set of web servers (say web servers 1-3). Still, they all shared the same back-end Blaise management server. Along the same lines, we have yet to explore what we could gain from having more than one server park on the same Blaise server. Finally, in addition to finishing our load-testing work in the first half of 2025, we envision transitioning all of our remaining production surveys to multi-server environments.

10. Conclusion

Load testing Blaise surveys is not for the faint of heart, as the process can have issues, and a steep learning curve exists. Ultimately, we recommend maximizing the performance of your production environments regardless of any empirical evidence from your internal load tests. The University of Michigan's first production study using a multi-server environment happened in 2024, where we had three front-end web servers and three back-end servers, where the Audit, Session, and Data roles were balanced across the servers. In 2025, we will have our first production: five web servers and one back-end server.

Through it all, we have adapted, and load testing is a journey. In the end, we are reminded of the song by Journey, *Don't Stop Believin'*; singing the chorus regularly:

*Don't stop believin'
Hold on to that feelin'
Streetlights, people...*

If nothing else, we hope this paper sparks conversations among teams to begin thinking about how you can optimize your Blaise server environments to achieve the best outcomes possible for your surveys and infrastructure.

Appendix A: Blaise install Command-line

Credential examples and obfuscation of other elements are listed in red text below.

Web server installation:

```
C:\Windows\SysWOW64\msiexec.exe /qr /I!* MsiExecLogs.txt /norestart /i
"d:\CurrentBlaise\Blaise5.msi" INSTALLATIONTYPE="Server" WEBSERVER=1
DATASERVER=0 DATAENTRYSERVER=1 RESOURCESERVER=1 SESSIONSERVER=0
AUDITTRAILSERVER=0 CATISERVER=0 DASHBOARDSEVER=0 EVENTSERVER=0
CARISERVER=0 PUBLISHSERVER=0 CASEMANAGEMENTSERVER=0
MACHINEKEY=PerformanceServersTestPerformanc SERIALNUMBER=AAAA-AAAAA-
AAAA-AAAAA-AAAA LICENSEE=IBUC ACTIVATIONCODE=AAAA-AAAA-AAAA-
AAAA FORCEINSTALL=1 MANAGEMENTBINDING=https EXTERNALBINDING=https
CERTIFICATESTORENAME=My CERTIFICATESTORELOCATION=LocalMachine
CERTIFICATETHUMBPRINT=92B79025944198420AF019643068AE129271D25
DELETEDSETTINGS=1 DELETERUNTIMEENVIRONMENT=1
```

Admin installation:

```
C:\Windows\SysWOW64\msiexec.exe /qr /I!* MsiExecLogs.txt /norestart /i
"d:\CurrentBlaise\Blaise5.msi" INSTALLATIONTYPE="Server" WEBSERVER=0
DATASERVER=1 DATAENTRYSERVER=0 RESOURCESERVER=0 SESSIONSERVER=1
AUDITTRAILSERVER=1 CATISERVER=1 DASHBOARDSEVER=1 CARISERVER=1
PUBLISHSERVER=1 CASEMANAGEMENTSERVER=1 EVENTSERVER=1
MACHINEKEY=PerformanceServersTestPerformanc SERIALNUMBER=AAAA-AAAAA-
AAAA-AAAAA-AAAA LICENSEE=IBUC ACTIVATIONCODE=AAAA-AAAA-AAAA-
AAAA SERVERPARK="TestPark" FORCEINSTALL=1 MANAGEMENTBINDING=https
EXTERNALBINDING=https CERTIFICATESTORENAME=My
CERTIFICATESTORELOCATION=LocalMachine
CERTIFICATETHUMBPRINT=B059E3A36F3BCCCECC7336DC0938ADF50FE533B9
DELETEDSETTINGS=1 DELETERUNTIMEENVIRONMENT=1
```

Server park creation (run from the admin server):

```
"C:\Program Files (x86)\StatNeth\Blaise5\Bin\ServerManager.exe" -addserverpark:TestPark -
server:perfserver4 -Binding:https -publicbinding:https -user:XXXXXXX -password:XXXXXXX
```

Adding each web servers to the server park (run from admin server):

```
"C:\Program Files (x86)\StatNeth\Blaise5\Bin\ServerManager.exe" -asps:perfserver1 -
server:perfserver4 -password:XXXXXXX -user:XXXXXXX -masterhostname:perfserver4 -
serverpark:TestPark -roles:WEB,DATAENTRY,RESOURCE -serverbinding:https -
publicbinding:https
```

```
"C:\Program Files (x86)\StatNeth\Blaise5\Bin\ServerManager.exe" -editlogicalroot:default -
binding:https -port:443
```

Check server park creation:

```
C:\Program Files (x86)\StatNeth\Blaise5\Bin\ServerManager.exe -lspss -server:perfserver4 -
binding:https -user:XXXXXXXX -password:XXXXXXXX -protocol:wcf
```

Four servers have an 'Active' status in the MST server park.

Creating and setting the session data interface and audit trail interface:

Place the files DataInterfaceCreator.exe and DataInterfaceCreator.exe.config in the
..\StatNeth\Blaise5\Bin folder.

```
"C:\Program Files (x86)\StatNeth\Blaise5\Bin\DataInterfaceCreator.exe" -G:Session -
B:"SessionSQLServer.bsdi" -O:"D:\Helpers" -A:SQL -C:"Data Source=xxx.xx.xx.xx;Initial
Catalog=load;Persist Security Info=True;User ID=XXXXXXXX;Password=XXXXXXXX" -
S:DotNETFrameworkDataProviderForSQLServer -P:Stream -T:true -D:true -Q:true
```

```
"C:\Program Files (x86)\StatNeth\Blaise5\Bin\ServerManager.exe" -editconfigurationsettings -
sessiondatainterface:"D:\Helpers\SessionSQLServer.bsdi"
```

```
"C:\Program Files (x86)\StatNeth\Blaise5\Bin\DataInterfaceCreator.exe" -G:Audittrail -
B:"AudittrailSQLServer.badi" -O:"D:\Helpers\AudittrailSQLServer" -A:SQL -C:"Data
Source=xxx.xx.xx.xx;Initial Catalog=load;Persist Security Info=True;User
ID=XXXXXXXX;Password=XXXXXXXX" -S:DotNETFrameworkDataProviderForSQLServer -
P:Stream -T:true -D:true -Q:true
```

```
"C:\Program Files (x86)\StatNeth\Blaise5\Bin\ServerManager.exe" -editconfigurationsettings -
Audittraildatainterface:"D:\Commands\badi\AudittrailSQLServer\AudittrailSQLServer.badi"
```

```
"C:\Program Files (x86)\StatNeth\Blaise5\Bin\ServerManager.exe" -restartservices
```

Appendix B: Load test setup at CBS

Environment:

- Four dedicated Blaise servers - Windows Server 2019
- One ADMIN server: Roles ADMIN, DATA, SESSION, AUDITTRAIL, CATI
- Three WEB servers: Roles WEB, DATAENTRY, RESOURCE
- HTTPS configuration
- WCF protocol
- One dedicated database server with Oracle, MySQL, PostgreSQL, and SQL Server - Windows Server 2019
- One JMeter client machine - Windows 10
- No load balancer
- Power plan set to 'High performance' on all machines

Internal Name	Audit Trail	CARI	Cati	Data	Data Entry	Event	Publish	Case Management	Resource	Session	Web	Status	Blaise Version
perfserver4	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Active	5.15.2.3872
perfserver1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Active	5.15.2.3872
perfserver2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Active	5.15.2.3872
perfserver3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Active	5.15.2.3872

Machines are equipped with:

- Intel Core i7-9700 CPU @3.00 GHz, 300 MHz, 8 cores
- 32GB RAM (since 11-2-2025, was 16 GB)
- SSD HD

Blaise Instrument:

- Production-like survey used at CBS
- 100+ pages
- Mid-level complexity in pages
- External-/lookup
- Time to fill out 20 minutes
- Audit trail level 'Field'

JMeter configuration:

- Run duration 40 minutes
- Ramp-up period: 15 minutes
- Total number of users/sessions 1800
 - Users per web server: 600

Setting up the environment:

The steps below are executed automatically within our testing environment with each test build. AT CBS Load test, we perform four load tests, 1 for each database: SQL Server, Oracle, MySQL, and PostgreSQL.

Each run starts with a 'clean slate', meaning:

- Uninstalling and installing Blaise in our dedicated server park
- Creating a server park with an HTTPS connection
- Preparing the environment with the data interface files like Session (bsdi), Audit Trail (badi,) and all bdix files connecting to the database (dropping and creating tables).
- Building the instrument and filling in the needed data
- Installing the instrument

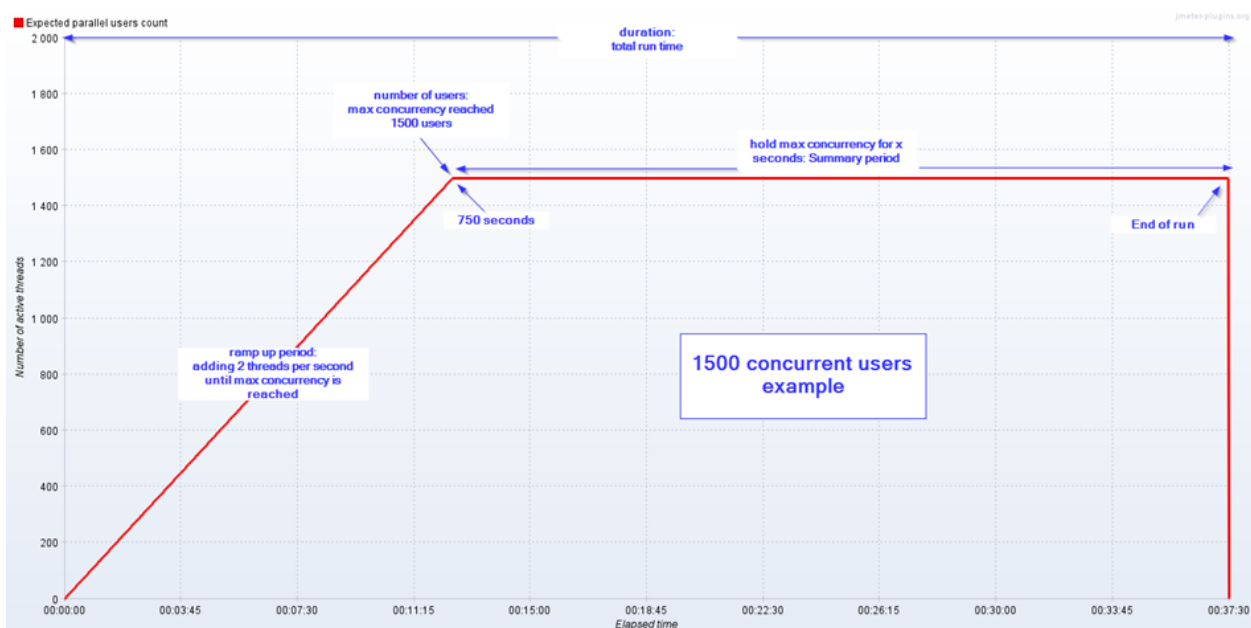
Once the server park environment is set up, JMeter will start a ‘warm-up’ from a dedicated client machine targeting each web server. One session (user) is started and all requests of the JMeter script are sent to the server until the receipt page is reached. This run is executed without pauses, so the timers in the requests are ignored. Once all three records are validated, the JMeter load test starts. We use 1800 concurrent users in our case, meaning each web server will handle 600 concurrent users.

After a run, we compare the results of the specific DB run with our baseline. JMeter run result example.

Label	# Samples	Average	Min	Max
executeaction	255494	79	22	3369
fetch_lookup_results	2178	86	11	402
select_external_record	2178	54	24	332
key_page	1811	139	59	394
receipt_page	1800	149	64	511
gezoexp	1800	126	46	3150
start_interview	1800	137	51	483
TOTAL	267061	80	11	3369

- **Label** is the request type.
- **Samples** is the number of requests sent to the server.
- **Average** is the average server response time for the specific request type and its requests.
- **Min** is the fastest server response time for the specific request type.
- **Max** is the slowest server response time for the specific request type.
- **TOTAL** is the average server response time for ALL request types and requests.

We use the JMeter Synthesis Report plugin to extract the logged response time after the ramp-up period. That’s when the maximum number of concurrent users is reached. We are interested in the results of the peak concurrency.



Baseline comparison

After a JMeter run for a specific database, we automatically compare the results of every request type with our baseline measurement. For every major release, we use the latest released build as a baseline; for instance, test build 5.14.9.3696 will be compared with baseline 5.14.8.3694.

For a new/in-development release, we use the latest major release version. For example, 5.15.1 baseline for 5.16 builds. There's an upper and lower deviation percentage norm where the results are still acceptable. If the results exceed the upper deviation, an error occurs, and we have to find out what went wrong. This could be a glitch in the environment or an issue within Blaise. A warning will be displayed if the results are lower than the lower deviation norm. This likely means the Blaise performance has improved due to code changes.

Time	Level	Category	Message
48:54.001	Info	Data	Current variable values: \$numberofusers = '1800', \$BaselineReleaseNr = '5.14.5', \$db = 'Oracle'
48:54.074	Success	Validate JMeter CSV file	executeaction baseline:83 and executeaction current:80 has an acceleration of -3,61% compared to baseline:83, an acceleration falls within the norm of 33%
48:54.120	Success	Validate JMeter CSV file	fetch_lookup_results baseline:34 and fetch_lookup_results current:25 has an acceleration of -26,47% compared to baseline:34, an acceleration falls within the norm of 33%
48:54.197	Success	Validate JMeter CSV file	gezoexp baseline:85 and gezoexp current:85 has no deviation and of 0% compared to baseline:85, no deviation and falls within the norm of 33%
48:54.245	Success	Validate JMeter CSV file	key_page baseline:139 and key_page current:144 has a delay of 3,6% compared to baseline:139, a delay falls within the norm of 33%
48:54.295	Success	Validate JMeter CSV file	receipt_page baseline:146 and receipt_page current:142 has an acceleration of -2,74% compared to baseline:146, an acceleration falls within the norm of 33%
48:54.346	Success	Validate JMeter CSV file	select_external_record baseline:57 and select_external_record current:55 has an acceleration of -3,51% compared to baseline:57, an acceleration falls within the norm of 33%
48:54.388	Success	Validate JMeter CSV file	start_interview baseline:144 and start_interview current:142 has an acceleration of -1,39% compared to baseline:144, an acceleration falls within the norm of 33%
48:54.420	Success	Validate JMeter CSV file	TOTAL baseline:83 and TOTAL current:80 has an acceleration of -3,61% compared to baseline:83, an acceleration falls within the norm of 33%

1800 users			
	5.13.19.3483	5.14.8.3694	5.15.1.3865
	Time (ms)	Time (ms)	Time (ms)
MySQL			
executeaction	201	80	84
fetch_lookup_results	212	87	88
gezoexp	197	119	97
key_page	277	152	157
receipt_page	321	151	162
select_external_record	207	55	56
start_interview	300	139	143
Oracle			
executeaction	201	83	80
fetch_lookup_results	159	28	25
gezoexp	168	114	85
key_page	265	138	144
receipt_page	296	142	142
select_external_record	207	56	55
start_interview	306	146	142
SQLServer			
executeaction	195	80	79
fetch_lookup_results	170	26	25
gezoexp	157	123	84
key_page	253	131	142
receipt_page	278	140	143
select_external_record	213	56	54
start_interview	286	141	140
PostgreSQL			
executeaction	188	81	82
fetch_lookup_results	150	26	27
gezoexp	171	117	90
key_page	252	133	144
receipt_page	280	142	146
select_external_record	199	55	56
start_interview	294	143	145

We also monitor all load test results between builds and releases:

MaxDeviation		33%											
MinDeviation		-33%											
				Version									
				5.12.18.3300 240918_0704		5.13.14.3473 240515_0154		5.14.8.3691 250109_2310		5.15.0.3856 241202_1513		5.15.1.3862 241216_1530	
				Rijlabels		Time (ms)		%Diff		Time (ms)		%Diff	
				1800									
				MySQL									
				executeaction		206		208		81		80	
				fetch_lookup_results		233		224		86		88	
				gezoexp		185		182		118		70	
				key_page		441		303		140		162	
				receipt_page		354		349		150		156	
				select_external_record		222		214		57		58	
				start_interview		343		336		142		144	
				TOTAL		210		211		83		81	
				Oracle									
				executeaction		193		210		81		81	
				fetch_lookup_results		150		200		25		25	
				gezoexp		170		189		110		73	
				key_page		300		272		137		148	
				receipt_page		297		314		141		147	
				select_external_record		207		231		55		56	
				start_interview		318		319		142		146	
				TOTAL		195		211		81		81	
				SQLServer									
				executeaction		203		192		78		77	
				fetch_lookup_results		188		157		25		26	
				gezoexp		189		153		132		69	
				key_page		381		270		129		140	
				receipt_page		320		297		136		140	
				select_external_record		221		200		52		54	
				start_interview		323		292		136		139	
				TOTAL		206		194		79		78	
				PostgreSQL									
				executeaction		190		222		79		82	
				fetch_lookup_results		154		210		26		34	
				gezoexp		153		190		106		70	
				key_page		335		299		131		149	
				receipt_page		276		332		133		143	
				select_external_record		198		244		54		58	
				start_interview		287		332		138		144	
				TOTAL		192		223		80		83	

Appendix C: Ensuring Unique Key Value per Thread in an Infinite Loop in JMeter

To maintain a unique key value per thread while running an infinite loop until the duration is reached, the following JMeter settings were applied:

- Track counter independently (in key value counter): OFF
- Same user on each iteration: OFF
- Specify thread lifetime: ON
- Loop count infinite (GUI): OFF (controlled via `-Jloops=-1` in the command line)

Each thread receives a new key value on every iteration with this configuration, ensuring continuous and unique session creation. The test maintains a stable 1800 concurrent users, while new sessions keep starting throughout the test duration. The key value counter manages key assignment, ensuring each thread gets a unique key value at the start of every iteration. This prevents session depletion and ensures proper workload distribution in performance testing.

The screenshot shows the configuration for a Thread Group in JMeter. The Name field is set to `$_P(threadGroupName,x_ConcurrentUsers)}`. The Comments field is empty. Under "Action to be taken after a Sampler error", the "Stop Test" radio button is selected. The "Thread Properties" section includes: "Number of Threads (users)" set to `$_P(numberofusers,3)`; "Ramp-up period (seconds)" set to `$_P(rampupperiod,1)`; "Loop Count" set to "Infinite" with a checkbox highlighted in yellow; "Same user on each iteration" checkbox highlighted in yellow and unchecked; "Delay Thread creation until needed" checkbox unchecked; "Specify Thread lifetime" checkbox checked and highlighted in yellow; "Duration (seconds)" set to `$_P(duration,0)`; and "Startup delay (seconds)" set to 0.

Counter

Name:

Comments:

Starting value:

Increment:

Maximum value:

Number format:

Exported Variable Name:

Track counter independently for each user

Reset counter on each Thread Group Iteration

Warmup Configuration for JMeter Load Test

To execute a warmup phase before the actual load test, the following JMeter settings were applied:

- Number of users: 3 (small-scale warmup)
- Ramp-up period: 1 second (almost instant start)
- Loop count: 1 (each thread runs a single iteration)
- Duration: 60 seconds (test stops after 1 minute)
- Constant Throughput Timer delays: OFF (ensuring no enforced pacing)
- Timer factor: 0 (no artificial delay between requests)
- Key value assignment: Starts from 18000
- Three web servers available (loaded from the CSV configuration)
- Each thread is assigned a unique web server (based on the CSV Data Set Config)

Since all delays are disabled, the test completes in approximately 10 seconds rather than running for the entire 60-second duration. The 60-second duration acts as a failsafe in case of unexpected delays or system issues, preventing JMeter from running indefinitely if something goes wrong.

This configuration ensures that each thread runs exactly once, allowing for a quick warmup to initialize systems and ensure stability before the full load test begins. The key value counter still manages key assignment, ensuring unique session initialization. Additionally, thanks to the CSV Data Set Config, each thread is dynamically assigned a unique web server from the provided list, simulating a distributed request pattern.

Command Used:

```
D:\JMeter\bin\jmeter.bat -f -n -t D:\JMeter\scripts\LoadTest.jmx -l
D:\JMeter\results\Warmup\Warmup_Report.csv -e -o D:\JMeter\results\Warmup -Jnumberofusers=3 -
Jrampupperiod=1 -Jloops=1 -Jduration=60 -Jconstant_throughput_timer.delays=false -Jtimer.factor=0 -
Jkeyvalue=18000
```

Key Execution Details (Warmup Run):

- **Test Script:** [LoadTestGezo.jmx](#)
- **Users:** 3 concurrent users, one for each web server
- **Ramp-up Period:** 1 second
- **Test Duration:** 60 seconds
- **Loops:** 1 loop per user
- **No Delays:** Timers are disabled (`-Jconstant_throughput_timer.delays=false -Jtimer.factor=0`)
- **Results File:** `.jtl` file for logging warm-up test results
- **Report:** An HTML report is generated for analysis

Parameter Explanations:

Parameter	Description
D:\JMeter\bin\jmeter.bat	Starts JMeter in non-GUI mode.
-f	Overwrites the results file if it already exists.
-n	Runs JMeter in non-GUI mode for command-line execution.
-t D:\JMeter\scripts\LoadTest.jmx	Specifies the JMeter test script (JMX file).
-l D:\JMeter\results\Warmup\Warmup_Report.csv	Saves test results to a CSV file.
-e -o D:\JMeter\results\Warmup	Generates a JMeter HTML report in the specified folder.
-Jnumberofusers=3	Runs the test with three concurrent users.
-Jrampupperiod=1	Sets a 1-second ramp-up period (users start immediately).
-Jloops=1	Each user runs only one loop (iteration).
-Jduration=60	Test runs for 60 seconds max, even if loops haven't finished. Since there are no artificial delays, this is more than enough time for one entire loop to complete.
-Jconstant_throughput_timer.delays=false	Disables any artificial delays from the Constant Throughput Timer.
-Jtimer.factor=0	Ensures no extra delays are applied by timers.
-Jkeyvalue=18000	Passes a key value parameter

Summary of JMeter Load Test Execution

This command runs a JMeter performance test in non-GUI mode with 1800 concurrent users, gradually ramping them up over 15 minutes (900s) and running the test for a total duration of 40 minutes (2400s). The results are saved in a JTL file and an HTML report is generated for analysis.

Command Used:

```
D:\JMeter\bin\jmeter.bat -f -n -t D:\JMeter\scripts\LoadTestGezo.jmx -l
D:\JMeter\Results\Run_1800_Gezo_Report_SQLServer.jtl -e -o
D:\JMeter\Results\Run_1800_Gezo_Report_SQLServer -Jkeyvalue=18003 -
Jnumberofusers=1800 -Jrampupperiod=900 -Jduration=2400 -
JthreadGroupName=1800_ConcurrentUsers
```

Key Execution Details:

- Test Script: [LoadTestGezo.jmx](#)
- Users: 1800 concurrent users
- Ramp-up Period: 900 seconds (15 minutes)
- Test Duration: 2400 seconds (40 minutes)
- Thread Group Name: [1800_ConcurrentUsers](#)
- Results File: [.jtl](#) file for logging raw test results
- Report: An HTML report is generated for analysis

Parameter Explanations:

Parameter	Description
jmeter.bat	Starts JMeter in non-GUI mode.
-f	Overwrites the results file if it already exists.
-n	Runs JMeter in non-GUI mode for command-line execution.
-t LoadTestGezo.jmx	Specifies the JMeter test script (JMX file).
-l Run_1800_Gezo_Report_SQLServer.jtl	Saves test results to a JTL file for further analysis.
-e -o Run_1800_Gezo_Report_SQLServer	Generates an HTML report in the specified folder.
-Jkeyvalue=18003	Passes a key value parameter (likely used within the script).
-Jnumberofusers=1800	Runs the test with 1800 concurrent users.
-Jrampupperiod=900	Gradually ramps up users over 900 seconds (15 minutes).
-Jduration=2400	Test runs for 2400 seconds (40 minutes) before stopping.
-JthreadGroupName=1800_ConcurrentUsers	Specifies the Thread Group Name used in the JMeter script.

Steps to Generate a Summary Report in JMeter using JMeterPluginsCMD

This guide explains generating a **summary report** from a JMeter test execution using JMeterPluginsCMD.bat.

Step 1: Ensure JMeter Plugins Are Installed

Ensure you have **JMeter Plugins** installed, as they provide the [SynthesisReport](#) feature.

- If not installed, download and install from [JMeter Plugins Manager](#).

Step 2: Run the JMeter Test and Generate a JTL File

Before generating a report, you need to have a JTL file containing test results.

Step 3: Generate the Summary Report

After the JMeter test run is complete, use the following command to create a summary report from the **JTL file**:

```
D:\JMeter\bin\JMeterPluginsCMD.bat --generate-csv "D:\JMeter\results\Summary.csv" -  
-input-jtl "D:\JMeter\results\LoadTest.jtl" --plugin-type SynthesisReport --include-labels  
gezoexp,runtime_parameters,json,start_interview,key_page,executeaction,fetch_lookup_r  
esults,select_external_record,receipt_page --start-offset 900
```

Explanation of Parameters:

- --generate-csv → Specifies the output CSV file for the report.
- --input-jtl → Specifies the JTL file containing JMeter test results.
- --plugin-type SynthesisReport → Uses the SynthesisReport plugin to summarize results.
- --include-labels → Filters the report to include only specific transaction labels.
- --start-offset 900 → Exclude the first 900 seconds from the summary to remove warm-up phase data.

Step 4: Verify the Report

- Open the generated CSV report at: D:\JMeter\results\Summary.csv
- Analyze response times, throughput, and errors.