Blaise 5 Scaling Experience – A Case Study

Mangal Subramanian, Kathleen O Reagan, Arthur Menis, and Ray Snowden, Westat

1. Introduction:

This paper describes Westat's experiences in implementing a large scale web survey using Blaise 5, focusing on the load testing and scaling part of the project. We also discuss various aspects of Blaise 5 as it relates to programming, server configuration, issues faced etc. The paper also touches on some alternative approaches to hosting Blaise 5 for such large scale projects, contrasting it to the one Westat used for the final implementation.

2. Project Background:

Westat conducted two simultaneous surveys (Main and Mode), using a common screener and separate extended surveys for eligible respondents, to provide a broad overview of the characteristics, attitudes, and experiences of two groups of respondents. The Main study was done entirely in WEB and a percentage of the Mode study was done in CATI and the remainder was done in WEB using the Main study's instrument. The mode study only allowed for only one of the two extended questionnaires.

Westat's role was focused on sampling, data collection, and weighting. Respondents were encouraged to complete the online screener and, if eligible, the appropriate online extended survey. The screener was offered in English plus two additional languages. Those who failed to respond were followed up with additional mailings that included a hard-copy instrument.

3. Blaise Instrument Characteristics:

The survey instrument was programmed in Blaise 5 (version 5.6.5.2055) as one instrument instead of 3 separate instruments in order to facilitate security measures which would allow respondents to go directly from the screener into a separate extended instrument without having to pass thru the management system repeatedly.

There were over 20 randomized questions programmed either by question order or category order in both extended instruments and 4 randomized question options in the screener. As we were using Blaise 5 version 5.6.5 we did not have the advantage of the randomization functions included in Blaise 5.7 so we randomized the "old fashioned way" with multiple arrays and voluminous code.

The client insisted on multiple questions on the same page but with individual templates which made movement from screen to screen sometimes noticeable. The survey instrument also included preloaded sampling random numbers and required sampling done in the screener.

4. Blaise web application performance requirements:

The total sample size for the project was over 350,000 and the initial plan was to send out all the mailers at the same time. So the estimates given by the project to the systems team were as follows:

- Initial estimate of 2,500 users per day accessing the survey,

- Approximate estimate of 400 concurrent users for the screener,
- 10% of the sample expected to qualify for the extended (ball park estimate of 30-40 concurrent users for the extended), and
- Acceptable page load time <= 3 seconds.

5. Application set-up:

The Blaise survey was managed by our in-house web survey management system. The role of the management system was to validate a pin entered by the user and launch the survey. At the completion of the survey the completion status was captured back into the management system.

Web security was an important requirement of the research. To ensure that the Blaise web session was secure from 'URL' hijacking or impersonation, the management system created a security token during launch and inserted it into a SQL table. When the Blaise instrument started, the token was checked to see if it was a valid one and the user was allowed to proceed only if it was valid.

6. Load Testing Process:

The load testing tool used was the Visual Studio Test Studio combined with the Azure Load testing service, which enables large number of users to be simulated through its cloud based tools, eliminating the need to set up and configure agents on premise.

6.1 Challenges with the Visual Studio Tool

We set up a couple of test scripts and ran them with a few users to validate that the tool is able to produce clean completes. Review of the Blaise DB results showed records were created, but no data were present that would have been expected based on the test scripts. We contacted the Blaise developers about this and they reported that the VS web test script worked with only the primary key as a script parameter. Blaise also has other data that changes from case to case; in particular the sessionID was most important, but there were others as well. The Blaise developers indicated that custom code would be needed to dynamically extract the sessionID that Blaise had generated on the server and substitute this value into later script requests to the server.

The Blaise developers offered a tool they used, but it did not work in our environment. So we reviewed our options and decided that the handling of a dynamic parameter such as sessionID was too involved to address within our limited schedule for setting up and testing a server configuration.

6.2 Implementing Load Testing

To mitigate the issue with load test tool, we decided to use a low-tech and less risky option – Manual Testing. We asked staff throughout Westat to access a test survey site and complete 5-6 user sessions in a span of 30 minutes. The final load test plan included the following elements:

- Over 70 staff completed the testing process;
- Each tester was provided with 5-6 unique URL's and asked to complete at least 3 sessions;
- Test scripts identified responses so that the routes chosen would be completed within the test period;
- The management application was eliminated from the load testing so the focus would be on the Blaise instrument, and a custom instrument was set up to accept the PIN directly; and

- Performance counters were set on the Web and Database servers to capture the findings.

7. Results:

Below we summarize some of the key metrics from the load test.

- A total of 333 completes were validated in the Blaise DB.
- This included mostly screeners, and a handful of extended interviews.
- The total test run time varied between 30 and 40 minutes.
- During this period the server monitors on both the DB and Web servers did not exhibit any abnormal behavior, sustained CPU spikes, or high memory usage.

This indicated that we had no memory leaks or other application performance issues. Therefore, we concluded that we had a successful process, and the manual load test helped us make a decision about the server configuration used for the project.

7.1 Interpreting the results

The following factors were considered in making a final decision for the project:

- With the manual load test we got 300 + completes in less than an hour with a single server configuration. The response times were less than 2 seconds for most pages except for one or two which had randomizations or complex layouts.
- The load test also had 70 users hitting the application hard, and trying to complete 4-5 sessions in 30 minutes; this probably would be less intense in actual production.
- From our past experience with load testing large web applications, the Database server was always a possible bottleneck and even if we had multiple application servers the DB would still be one.

So based on the above factors the final server configuration we came up with included:

- Two virtual web servers,
- One virtual database server, and
- Network load balancing using 'Kemp'.

7.2 Issues Found During the Load Test

During load testing, we did encounter a load time issue. A page that was reported as being slow by all users (10-11 seconds to load) was a transition from a question that had several randomizations. After consulting with the Blaise developers, we learned that the additional load time was due to the client rendering time (in this case a browser). And that with a large table the server time will increase some as it has to build a bigger page definition, while the client-side rendering will increase much more in these circumstances.

Our slow page had a layout that involved a group with seven enumerated type questions. We concluded that the problem was due to a client preference for separate questions, each with its own template, rather than a table. Although we do not have benchmarks for page loading with different layouts, we found manual testing to be very helpful in identifying these types of issues.

8. Production set up:

The production servers were each configured with a serverpark with all the roles: Management, Audittrail, CATI, Data, DataEntry, Resource, and Session. A local server (logical server) was created to allow the serverpark to be mapped to the website and the physical location of the installed survey. Figure 1 and 2 (below) shows the web and database server set-up and configuration, and Table 1 provides the server specifications.

Figure 1. Web and Database Server Set-Up

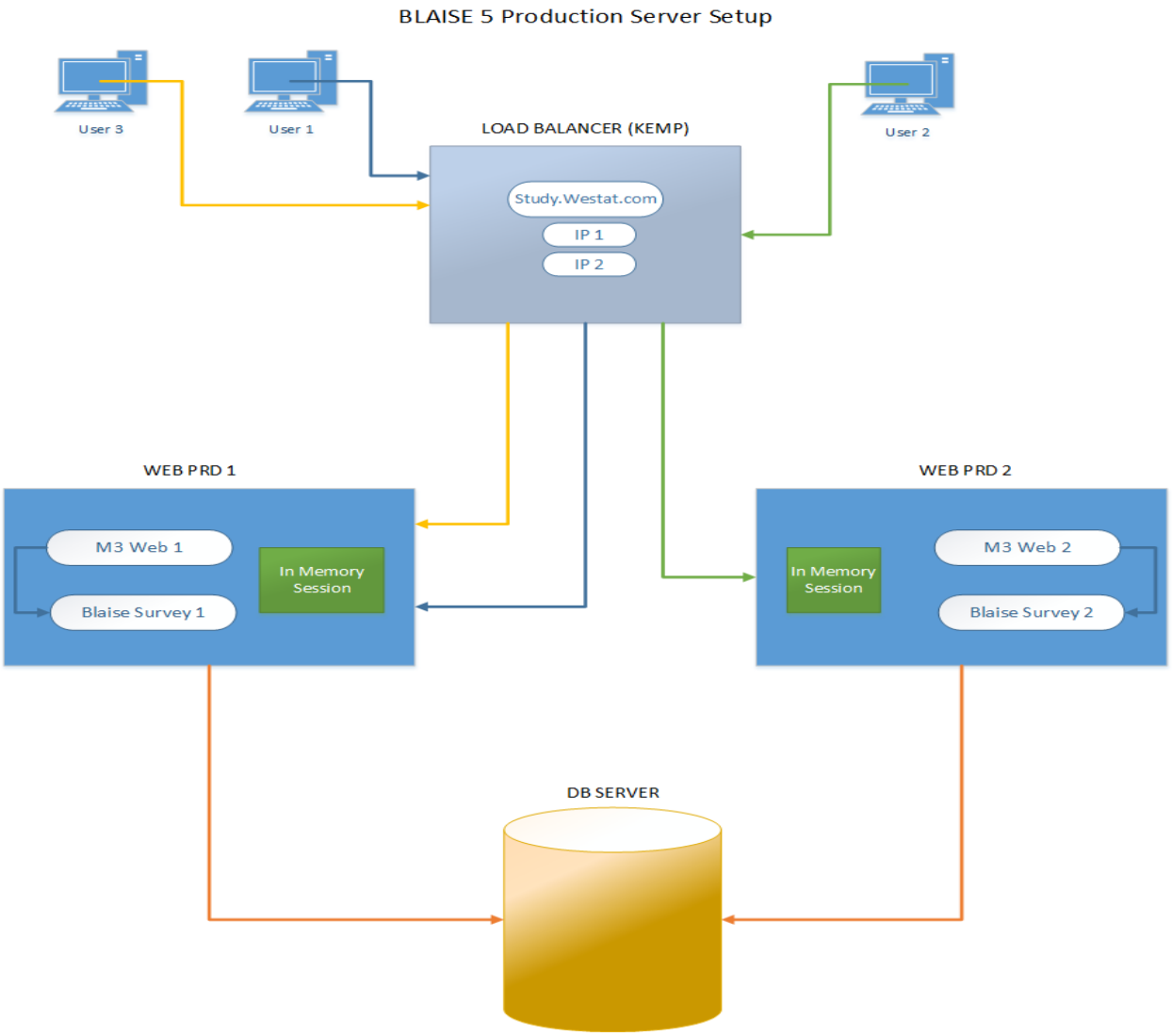
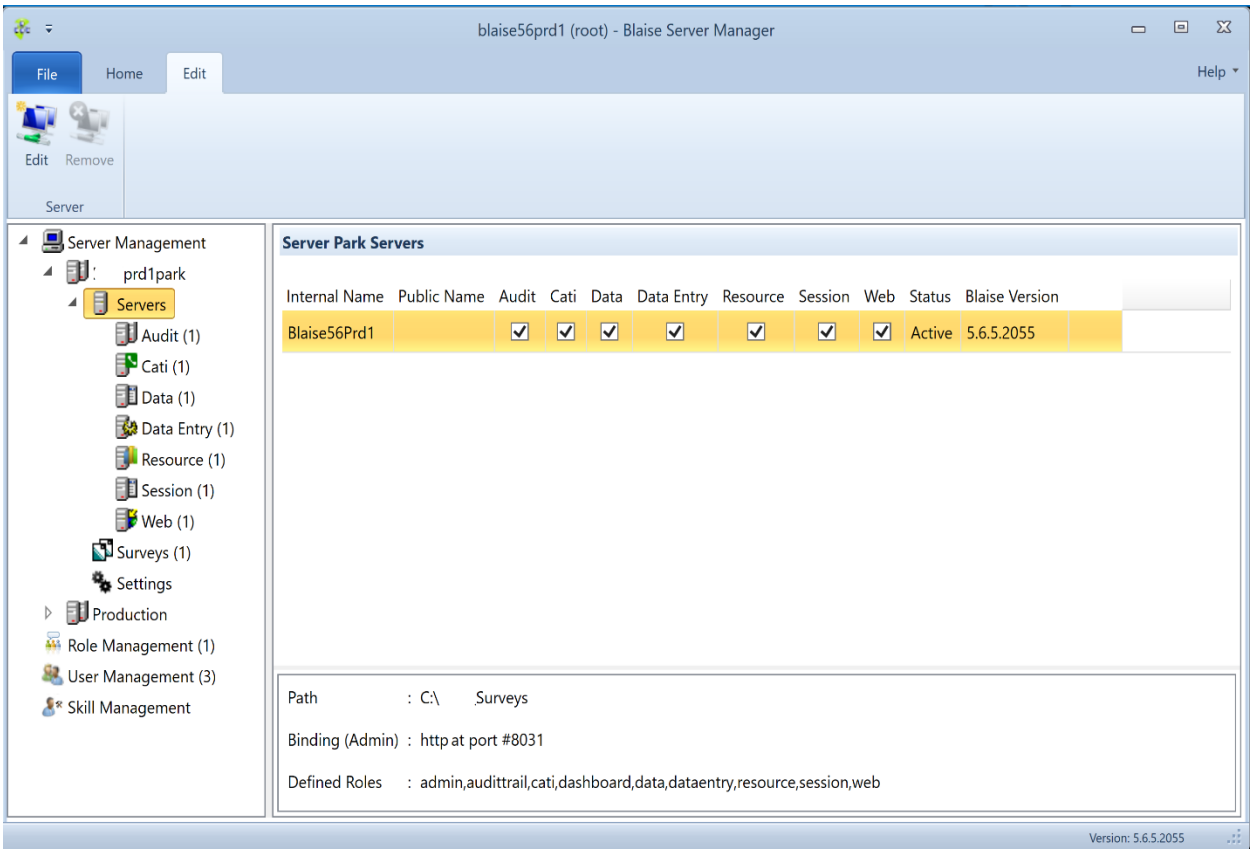


Table 1. Server Specifications

Server Type (virtual)	OS Version	CPU	Memory
DB Server	Windows 2016	Quad Core	32GB
Web Server 1	Windows 2016	Quad Core	32GB
Web Server 2	Windows 2016	Quad Core	32GB

Figure 2. Blaise Server Manager Configuration



9. Production Validation:

Once we had the production environment set-up we had to integrate the management system. We also conducted a small-scale load test to ensure that the production server set up with the load balancer was working smoothly. We report some observations below.

- The security implementation for the instrument involved Blaise doing a check with the token that was created by the management system. Since this was done using a 'Blaise' look-up and not a SQL read, there was a concern about the speed of doing the validation. This lookup was done by a Blaise search (TokenModel.SEARCH(pToken)) followed by a Blaise Read (TokenModel.READ). The lookup table itself was a SQL Server table with a primary key where keyvalues are GUIDs. In order to check whether this look-up would be a performance issue when we had 100k + tokens in the DB, we prefilled the table with two hundred thousand

(200,000) records and manually tested running cases. No perceptible slowness in response time was found so we concluded that the Blaise Read performed very well.

- Our initial smoke testing in production was not successful. Some sessions worked fine while others failed. On further analysis of the set-up, we realized that the management system to launch the instrument was set-up on only one server and we had to replicate it on the second server.

10. Performance in the field:

The project went live in early 2020, and overall performed really well, with very few issues reported through the help desk. Some issues that we did encounter are detailed below.

- *Session Data Overwritten*: The 'session data' for a case exists for two devices for the same case at the same time. When a respondent started the interview on one device (phone), but shortly thereafter decided to do the interview on a desktop, the phone session did not timeout until after the desktop session was completed. Because the respondent went to a device with a different ip-Address, the load-balancer didn't know to send the desktop case to the same server that the phone had been on. The respondent did successfully complete the case on the desktop. However, although the phone's 'session data' really didn't have much data, the phone's data overwrote (wiped out) the data collected on the desktop since it was the last to execute. This was fixed by looking at audittrails and re-entering the data.

- *Submit button click issue*: When a respondent reached the last page, but did not click the submit button (instead perhaps closing the browser to end the session), the case status and associated variables such as the timestamp were not set correctly. The submit button does some AssignField actions to set status and timestamp variables. This is something to be aware of when reporting on case completion.

- Metrics related to completions during peak user loads compared to an average of 496 completes to date. The maximum completes per hour during this peak load week was 197 cases.

Table 2. Peak load following project mailings in early January

Day of Week	Started	Completes
Monday	4,225	2,250
Tuesday	3,850	2,175
Wednesday	3,100	1,650
Thursday	4,350	2,300
Friday	4,400	2,400
Saturday	3,400	1,850
Sunday	2,500	1,275

11. Alternate Server Configurations:

Blaise 5's inherent architecture that consists of multiple server roles, lends itself to a distributed server farm set up to handle large-scale applications. The recommended server set up from Statistics Netherlands is represented below.

- Application Server 1 - Session, Data, Resource, Management, Audit Trail Web and DataEntry Roles.
- Application Server 2 - Web and DataEntry Roles
- Database server – That hosts the SQL DB.

Since we had no experience working with this type of configuration and the project was on a tight deadline, we did not have the time to set-up and test this. But we plan to test this server set up and to replicate our load tests.

12. Conclusion:

Based on our experience we believe that the Blaise 5 architecture is capable of handling large-scale web applications. Though we could not leverage the improvements in randomization functions in Blaise 5.7 due to the timing of this project's schedule, the web survey performed well in production with very few issues. There is still more work to do in the area of web load testing tools suitable for Blaise 5 and we are actively exploring multiple options.

Every project will have some unique characteristics that may require some tweaks and adjustments. This case study shows the powerful capabilities of Blaise 5 for large-scale web surveys.