Microsoft Office Project Server 2007 Performance Testing White Paper

May 2009

**Copyright**

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

 2009 Microsoft Corporation. All rights reserved.

Microsoft, the Office logo, SharePoint, SQL Server, and Windows are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

**Table of Contents**

[Introduction 5](#_Toc229464829)

[Pre-Reading Requirements 6](#_Toc229464830)

[Content Overview 6](#_Toc229464831)

[Sample Code 6](#_Toc229464832)

[Running a Project Server 2007 Performance Test Lab 7](#_Toc229464833)

[Planning 7](#_Toc229464834)

[Key Metrics for Performance 9](#_Toc229464835)

[Throughput 9](#_Toc229464836)

[Response Times 10](#_Toc229464837)

[Setting up the Environment 10](#_Toc229464838)

[Required Software and Configuration 10](#_Toc229464839)

[Data Population 10](#_Toc229464840)

[Generating XML files Supporting the Load Tests 10](#_Toc229464841)

[Testing User-initiated Read-only Operations Involving Project Web Access 11](#_Toc229464842)

[Access PWA Home Page 11](#_Toc229464843)

[Access Project Center Views 12](#_Toc229464844)

[Access Resource Center Views 12](#_Toc229464845)

[Access the My Tasks Page 12](#_Toc229464846)

[Access the My Timesheets Page 12](#_Toc229464847)

[Testing User-initiated Write Operations Involving Project Web Access 12](#_Toc229464848)

[Submit Task Updates 13](#_Toc229464849)

[Submit Timesheets 13](#_Toc229464850)

[Approve Task Updates 13](#_Toc229464851)

[Approve Timesheets 14](#_Toc229464852)

[Testing User-initiated Operations Involving Project Professional 14](#_Toc229464853)

[Open Projects 15](#_Toc229464854)

[Save Projects 15](#_Toc229464855)

[Publish Projects 15](#_Toc229464856)

[Testing Background Operations 15](#_Toc229464857)

[Cube Building 16](#_Toc229464858)

[Active Directory Synchronization 16](#_Toc229464859)

[Project Archiving 17](#_Toc229464860)

[Identify Different Queue Behaviors 17](#_Toc229464861)

[Extracting and Analyzing Test Data 18](#_Toc229464862)

[Moving Data from the Lab to the Test Analysis Database 18](#_Toc229464863)

[Extracting Relevant Data from the Test Database 18](#_Toc229464864)

[Test Environment 20](#_Toc229464865)

[Test Results 22](#_Toc229464866)

[Data Profiles 22](#_Toc229464867)

[Single Operation Load Tests 22](#_Toc229464868)

[Access PWA Home Page 23](#_Toc229464869)

[Access Project Center Views 25](#_Toc229464870)

[Access Resource Center Views 27](#_Toc229464871)

[Submit Task Updates 29](#_Toc229464872)

[Submit Timesheets 32](#_Toc229464873)

[Approve Task Updates 36](#_Toc229464874)

[Approve Timesheets 38](#_Toc229464875)

[Open Projects 42](#_Toc229464876)

[Save Projects 44](#_Toc229464877)

[Publish Projects 46](#_Toc229464878)

[Mixed Load Tests 50](#_Toc229464879)

[Testing Against Multiple Data Profiles 51](#_Toc229464880)

[Testing Against Multiple Server Topologies 51](#_Toc229464881)

# Introduction

The direct goal for Performance Testing is generally to determine how fast a system performs under a particular workload. However, Performance Testing can also be aimed to verify other attributes of the system, like scalability, reliability under stress, and resource usage patterns. Finally, Performance Testing results are often used as an input or a verification for capacity planning and system sizing.

The scenarios mentioned above are only representative of many other possible specific goals that can drive and motivate a Performance Testing effort. That’s why it is fundamental, before starting any activity, to identify and agree with all the stakeholders on the specific questions that you want your Performance Testing results to answer. Another question added later, when your test lab is not available anymore, is not likely to find an answer.

For more information about performance testing, see [Performance tuning lab on TechNet](http://go.microsoft.com/fwlink/?LinkId=150595) (http://go.microsoft.com/fwlink/?LinkId=150595).

When it comes to Performance Testing, Microsoft Office Project Server 2007 has several peculiarities that are worth mentioning here as they are the basic reasons for this specific white paper to exist:

* Office Project Server 2007 is not purely a web application. The interaction of Project Professional with the server follows the smart client pattern, and the Project Web Access (PWA) web application still has some ActiveX components (the Project Center and Resource Center grids) that run client-side in the browser. Client-side execution increases complexity in load generation and performance testing.
* Project Server can be used in so many different ways by different organizations. Things like feature areas in use, usage patterns, data profiles, specific configurations and/or customizations etc., can all have a very significant impact on the testing of a Project Server deployment, and it’s not practically feasible to test every possible combination of all these variables. For this reason, there can’t be any “pre-cooked” recipe for performance testing of a Project Server deployment, and this document should therefore only be used as a reference set of information and guidelines rather than as a prescriptive guidance.
* Usage patterns in Project Server 2007 are often articulated in complex work flows. As an example, if you are concerned about the performance of time tracking, you may want to include in your test cycles several different activities starting from timesheet save and submit, then timesheet approval, import in My Tasks, up to task update approval. This adds complexity to the test design because subsequent operations depend on the completion of previous tasks.
* The queuing sub-system in Project Server 2007 is deeply involved in most of the write operations. The main design goal for the queuing sub-system in Project Server 2007 is to smooth peak periods across longer periods of time, without forcing users to wait for processing completion. A consequence of such a behavior is that you don’t need to size the system strictly against peak periods, as long as you can accept longer times to full processing completion for write operations during peaks. It’s easily understandable that the queuing sub-system is one of the most complex factors to take into consideration when running a performance test lab, as well as when planning for capacity of a Project Server 2007 deployment. For this reason, you should always take particular care with the queues in both the design and execution phases for your tests. For example, you should always start a test run with the queues completely empty, and you should always include the queue draining time in your test runs.
* Project Server 2007 automatically manages a number of background activities, directly or indirectly related to user activities. Typical examples are the reporting synchronization jobs or the notification processing jobs. Performance tests should take into consideration the load generated by these background operations as they can affect resource utilization on the servers.

## Pre-Reading Requirements

Readers should have a deep functional and technical understanding of the Microsoft Enterprise Project Management solution based on Project Server 2007 before reading this document. For a comprehensive list of available documentation about Project Server 2007, see the Project Server 2007 TechCenter on TechNet at <http://technet.microsoft.com/en-us/office/projectserver/default.aspx>.

Other specific resources that should be familiar before reading this document are:

* “Performance Testing Guidelines for Web Applications” P&P book (<http://msdn.microsoft.com/en-us/library/bb924375.aspx>)
* Performance Testing Guidance (<http://www.codeplex.com/PerfTesting>)

## Content Overview

The first chapter (“Running a Project Server 2007 Performance Test Lab”), describes how to run performance tests against a Project Server 2007 deployment by using Microsoft Visual Studio Team System 2008 and a set of community-based tools built for the purpose.

The second chapter (“Test Environment”), documents the specifics of the lab environment which was used for the tests we ran for collecting data for this white paper.

The third chapter (“Test Results”), describes in detail the test scenarios that were conducted, and summarizes the data collected for every scenario. As discussed above, the reader should keep in mind that even though we tried to test against the most common scenarios, usage patterns and data profiles, every specific implementation has its own characteristics and can differ, even substantially, from the results we describe here.

## Sample Code

This document includes references to sample code included in the “Project Server 2007 Performance Lab Kit”, which is available at: <http://code.msdn.microsoft.com/ps07PLT>.

To learn more about the Project Server programming interface, refer to the Project Server 2007 Software Development Kit on MSDN: <http://www.microsoft.com/downloads/details.aspx?familyid=2672F6F9-7028-4B30-99A2-18CB1EED1ABE&displaylang=en>

# Running a Project Server 2007 Performance Test Lab

Running a performance test lab against an Office Project Server 2007 solution can be quite a complex task, due to the variety of user interactions and system behaviors involved. The following are the main types of load that we normally need to reproduce in a performance test environment:

* User-initiated operations involving Project Web Access
* User-initiated operations involving Project Professional
* Background operations

Testing strategies can include independent loads or mixed loads composed of different operations. Measuring how the systems react to the load produced by a single specific operation by many simulated users at the same time can help determine a reference measure for peak periods of that specific operation, while mixed loads strategies are more realistic and can be used to simulate how the systems react to several possible usage patterns. For example, if you are concerned about the timesheet submission load, which in your specific scenario has a peak period on Friday afternoon for all of your users, then you can test the system by simulating timesheet submissions only, without any other simulated load. This will provide a reference baseline in terms of maximum throughput (e.g., how many timesheets can be submitted/processed per hour) of your system for the particular operation under investigation. However, you may also want to test the system under a combination of a wider set of different operations at the same time, using usage profiles as similar as possible to your real scenario, ideally using historical logs coming from the production system as a data source to determine the usage profiles to be tested.

Regardless of the overall testing strategies you want to adopt, the following paragraphs describe in more detail how to plan and simulate the load types belonging to the three categories above. Visual Studio Team System 2008 Test Edition provides the foundation for all the scenarios described below. Using Visual Studio Team System 2008 Test Edition, you can easily balance multiple load types in one single load test to run for a given period of time.

For more information about Visual Studio Team System, see the Team System Development Center on MSDN (http://msdn.microsoft.com/en-us/teamsystem/default.aspx).

## Planning

Planning is the first and most important phase of every complex task, and performance testing is not an exception to this rule. For specific information about planning your Office Project Server 2007 deployment, see "Planning and architecture for Office Project Server 2007" on TechNet (http://technet.microsoft.com/en-us/library/cc197605.aspx).

Before setting up the environment and running tests, you should thoroughly plan for all the aspects of what you are going to do. The following table summarizes some of the key points you need to plan for.

| Topic | Description |
| --- | --- |
| Hardware | Your lab configuration should be as close as possible to your existing or target production environment.  Particular attention should be put on the choice between 32-bit and 64-bit architectures. 64-bit architectures are strongly preferable, but if you are constrained to 32-bit in your production environment, your lab environment should reflect the same configuration.  Ensure you keep track of the details of your hardware configuration before dismissing the lab, for future reference. |
| Software | Plan for installing the latest fixes available for every software component in your lab. If you plan to run the lab across a significant amount of time (one month or more), you should also plan for updating the systems when needed with the latest security fixes. If not strictly needed, you should minimize other changes to the software configuration of your systems during the lab timeframe, in order to maintain data comparability across different test runs performed at different times. |
| Storage | Your lab should have enough storage space to store:   * Live data   + Project Server databases   + SharePoint Server databases * Backups   + One backup set right after the lab setup   + Backup sets of Project Server and relevant SharePoint Server databases for every different data profile you need to use * Test results   + The storage size required for every test run depends on the set of data you collect, the sampling rate, and the test length |
| Network | Your lab environment should be placed on an isolated network, in order to minimize the impact of extraneous network activities on your tests. One machine in the lab is normally configured as a bridge for remote access to the lab from the corporate network. |
| Directory Services | Since you will need to simulate users accessing your lab environment, and you need to know the passwords for each of the simulated users, you need to plan for the Directory Services to be used. You should plan for a dedicated Organizational Unit (OU) in an existing Active Directory directory service domain for managing the test user accounts, or for a dedicated Active Directory domain for your lab environment. |
| Test Scenarios | Depending on your overall goals for the performance lab, your test scenarios will need to be planned carefully. Consider the following elements for every test scenario:   * Operations to be simulated (either a single operation or a mixed set of operations with different percentages) * Users and roles that need to be simulated for every operation (e.g., how many different users for every role involved, etc.) * Data profile to be created at the beginning of the test (e.g., how many projects, tasks per project, assignments per resource, assignments per project, etc.) * Test duration and load pattern (e.g., warm-up time, step load, etc.) * Data to be collected (e.g., which counters from which servers, sampling rate, etc.) * Acceptance criteria and thresholds (e.g., less than *x*% errors, avg. CPU utilization < 60%, etc.) |
| Data Profiles | Putting together all the test scenarios you will perform, plan for your overall data population strategy by identifying the minimum number of data profiles you will need for your tests. Data profiles typically include the following elements:   * Users   + User accounts   + Resource Breakdown Structure (RBS)   + Enterprise Resource Pool   + Project Managers and other roles   + Security (authentication mode, groups, categories) * Enterprise custom fields * Projects   + Tasks   + Task dependencies   + Team   + Assignments   You should plan for an appropriate naming convention for all of your test entities (users, projects, tasks, etc.). |

## Key Metrics for Performance

Throughput and response times are two common metrics for measuring required, expected or actual performance of a system. The following two sections define these two metrics, as they are used heavily throughout the document.

### Throughput

*Throughput* is a measure of the number of operations the system can handle in a unit of time. Throughput is typically expressed in operations per second, but you need to clearly identify what an “operation” is in every specific context. For example if you think of a web page, you can refer to one operation as the serving of a whole page, or every single http request the server receives to serve the page (a web page can contain images and other resources that are requested independently). The two possible different definitions in the example above should clarify why you need to get or provide clarity about what an “operation” is, whenever you deal with a throughput measure.

Estimating the required throughput for a system is a challenge that requires a deep understanding of the usage patterns of your users. An industry average suggests that 1 operation per second maps to 1,000 users, based on the following calculation:

* 1,000 users work on average at 10 percent concurrency.
* Therefore, on average there are 100 simultaneous users on a 1,000-user system.
* For each of the 100 simultaneous users, there are 100 seconds per operation per each user (the user “think time”).
* If an active user pauses 100 seconds between operations, then the user will generate 36 operations per hour (3,600 seconds in an hour divided by 100 seconds between user requests equals 36 operations generated by the user).
* If users average 36 operations per hour, and there are 100 simultaneous users, the simultaneous users will request on average a total of 3,600 operations per hour. As there are 3,600 seconds in an hour, users will require a solution that can provide 1 operation per second (3,600 seconds per hour / 3,600 user operations per hour).

Of course, the assumptions of the above calculation should be adapted to your specific scenario in terms of user concurrency, peak factors and usage patterns.

Anyway, a throughput of 10 operations per second doesn’t mean that every operation is fully processed in 0.1 s, but only that the system is handling 10 operations in that second. That’s why the “response time” is a separate metric, as important as throughput in terms of performance.

### Response Times

Independently of how many operations the system is capable of managing concurrently, another measure of performance which is even more important to users is absolute *response times*. Response time degradation can be a good indicator of capacity issues. There are a range of potential response time bottlenecks, such as disk access, network I/O, memory, or processor problems.

Response times depend significantly on several factors like operation types, data profiles, systems configuration, etc. It’s also fundamental that you define in detail the acceptance thresholds in response times for all the different operations you’re considering.

## Setting up the Environment

### Required Software and Configuration

In addition to the servers composing the Project Server 2007 farm and the needed infrastructure services like domain controllers, etc., you will need to set up a Visual Studio Team System 2008 Test Edition controller – agent topology. See <http://msdn.microsoft.com/en-us/library/ms182634.aspx> for more information on Visual Studio Team System 2008 Test Edition. It’s strongly recommended that you use dedicated hardware for the controller, the agent(s) and the database used to store test results, in order to minimize the impact on systems under test.

Similarly, in order to test operations involving Project Professional, you will also need to set up a test infrastructure capable of automating client operations across multiple Terminal Services sessions, and controlling test scenarios in a centralized manner. An example of such a test solution is the community-based project called Test Framework, which is available at <http://www.codeplex.com/TestFramework>.

Right after the setup completion of your test environment, it’s a good choice to perform a full backup of everything, and save the backup set so that you’ll always be able to jump back to the baseline state in case of need.

### Data Population

Once your Project Server 2007 farm is up and running, you can start generating the data profiles you planned for. One tool you can use to assist in this process is the community-based solution called EPM 2007 Test Data Population Tool, which is available at <http://www.codeplex.com/EPMDataPop>.

After you complete the generation of one specific data profile, perform a full backup of the Project Server databases[[1]](#footnote-2) (and the SharePoint content database if you include Project Workspaces in your test scenarios) and save the backup set so that it will be available for restore every time you need to start a test run with that data profile. It’s important that every time you start a new test run you perform a restore of the backup set containing the appropriate data profile, so that you can ensure the same initial conditions of multiple test runs for the same scenario.

### Generating XML files Supporting the Load Tests

Load tests will need to impersonate a number of different users in order to provide the most accurate simulation of reality. This can be accomplished by using the data binding feature in Visual Studio Team System 2008 Test Edition, so that every test instance will randomly pick one user from the whole list. The same approach can be taken for binding other data to your coded tests, like for example project names, resource UIDs, etc.

The “EPMTestTools” Visual Studio solution, which is part of the “Project Server 2007 Performance Lab Kit” (see “Sample code” above), provides a sample that can be used to enumerate all the resources from the Enterprise Resource Pool and to generate XML files containing key information that will be made available to the tests.

## Testing User-initiated Read-only Operations Involving Project Web Access

This paragraph refers to read­-only operations that users can initiate by using the Project Web Access application only, without the involvement of Project Professional on the client computer. Some of the most frequent operations in this category you may want to test include the following:

* Access the PWA home page
* Access Project Center views
* Access Resource Center views
* Access the My Tasks page
* Access the My Timesheets page
* Access Data Analysis views
* Access project workspaces

To simulate the above types of activity in a lab environment, you can easily create the corresponding web tests in Visual Studio Team System 2008 Test Edition. The following paragraphs provide specific examples about how to build the tests in Visual Studio 2008 for some of the aforementioned operations.

### Access PWA Home Page

You can use the Web Test Recorder tool available with Visual Studio 2008 to record all the http requests occurring when a user accesses the PWA home page.

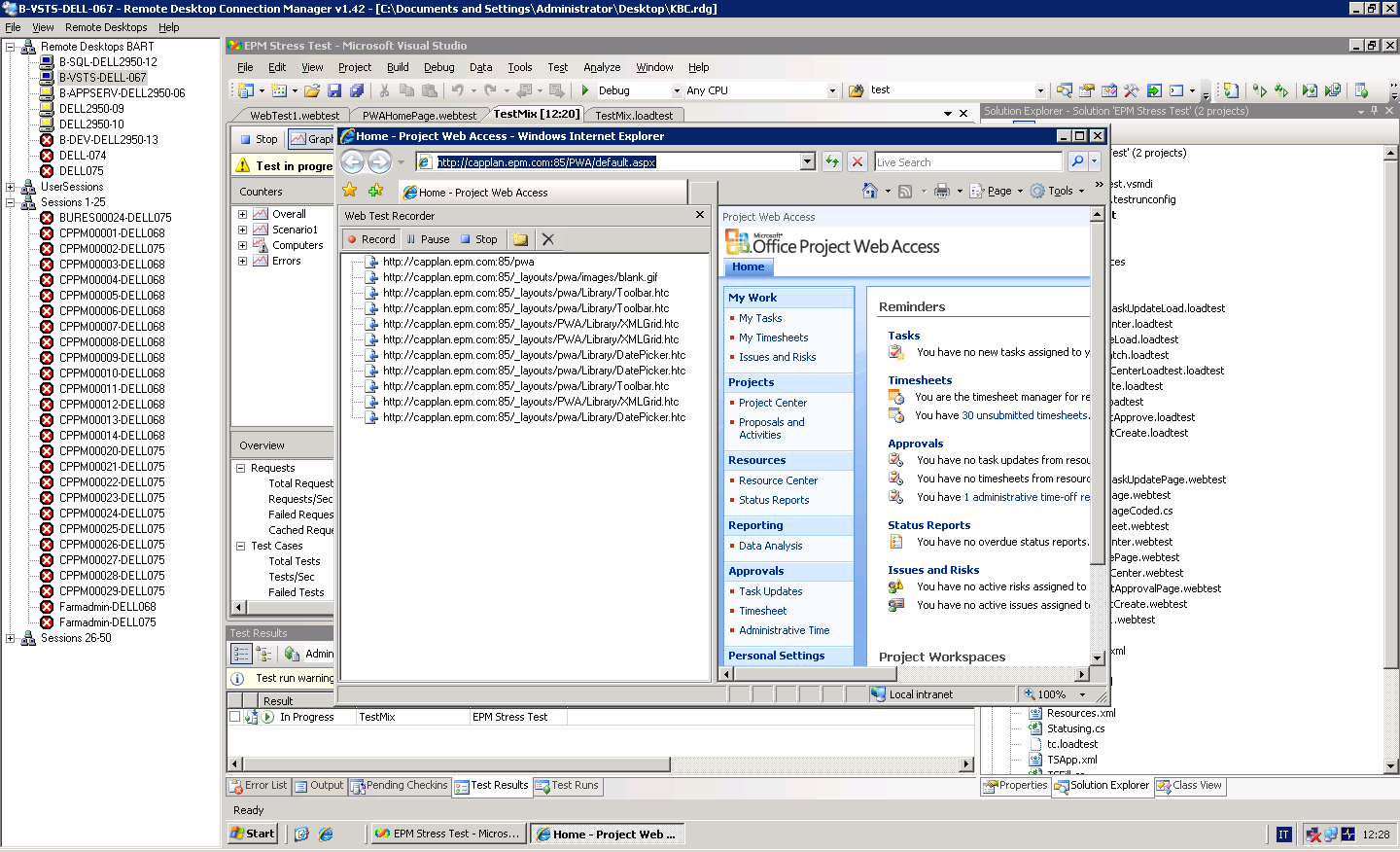


Figure 1 – The Visual Studio 2008 Web Test Recorder interface on the PWA home page

Once you have recorded the web test, you need to set up the credentials in order to simulate different users in the load scenario, and let Visual Studio randomly pick a user from a list for every test run. You can bind user credentials to a data source, which can read data from a database, an XML file, a CSV file or any other possible data source that is more convenient in your scenario. As an example, you can use the resources.xml file described above (see “Generating XML files supporting the Load Tests”) as the data source for user credentials. The same approach can be used for all the other tests as well.

### Access Project Center Views

Similarly to the PWA home page test, you can simply use the Web Test Recorder to record all the http requests occurring when a user accesses the Project Center. The Project Center page contains an ActiveX control, used to render the grid view. The Web Test Recorder will record the requests originated by the ActiveX control.

Multiple occurrences of pwa.asmx requests represent four separate calls to the PWA web service, all originated from the ActiveX control.

It’s worth mentioning that the default Project Center view for the simulated user will be the one used in the test.

### Access Resource Center Views

The Resource Center test is very similar to the Project Center test, since it also contains the Grid ActiveX control.

### Access the My Tasks Page

Again, using the Web Recorder tool in Visual Studio is enough to build a test for the My Tasks page. The number of occurrences for DatePicker.htc and blank.gif requests can vary based on the actual data showed for the simulated user. Since the web test is static, you should record a different web test for every different data profile you have so that the right number of requests will be performed. An alternative approach would be to use a coded test to manage the number of occurrences programmatically.

### Access the My Timesheets Page

The Web Recorder tool in Visual Studio will again suffice in building a test for the My Timesheets page.

## Testing User-initiated Write Operations Involving Project Web Access

This paragraph refers to write operations that users can initiate by using the Project Web Access application only, without the involvement of Project Professional on the client computer. Some of the most frequent operations in this category you may want to test include the following:

* Update status information on My Tasks
* Submit a timesheet
* Approve task updates
* Approve timesheets

While web tests alone are enough for read-only operations, for write operations some custom code leveraging the PSI will be required.

When defining the test mixes, you should always consider one read-only test corresponding to every write test. For example, in order to submit timesheets a user will always need to access the My Timesheets page.

### Submit Task Updates

The “EPM Stress Test” Visual Studio solution, which is part of the “Project Server 2007 Performance Lab Kit” (see Sample code paragraph above), provides a sample test method (“SubmitTaskUpdates”) to implement the Submit Task Updates test. The sample code picks a random assignment from a list of all the assignments in the system (pre-generated in an xml file), and then impersonates the resource assigned to that assignment and submits a change of the progress (percent complete).

In order to test task update operations realistically, however, you also need to add a web test to hit the My Tasks page. You can simply set up a web test for the My Tasks page by using the Web Test Recorder in Visual Studio, as described above. The test mix between the web test (page hit) and the unit test described above (actual task update submission) should be set as follows (two page hits for every submission):

|  |  |  |
| --- | --- | --- |
| Operation | Test | Percentage |
| Submit Task Updates | MyTasksPage web test | 67 |
| SubmitTaskUpdates unit test | 33 |

### Submit Timesheets

The “EPM Stress Test” Visual Studio solution, which is part of the “Project Server 2007 Performance Lab Kit” (see "Sample code" paragraph above), provides a sample test method (“CreateAndSubmitTimeSheet”) to implement the Submit Timesheets test. The sample code picks a random resource and a random period, creates the timesheet if not already existing, fills the timesheet randomly, saves the timesheet and finally submits it.

In order to test timesheet submission operations realistically, however, you also need to add a web test to hit the My Timesheets page. You can simply set up a web test for the My Timesheets page by using the Web Test Recorder in Visual Studio, as described above. The test mix between the web test (page hit) and the unit test described above (actual timesheet submission) should be set as follows (two page hits for every submission):

|  |  |  |
| --- | --- | --- |
| Operation | Test | Percentage |
| Submit Timesheets | MyTimesheet web test | 67 |
| CreateAndSubmitTimeSheet unit test | 33 |

### Approve Task Updates

The “EPM Stress Test” Visual Studio solution, which is part of the “Project Server 2007 Performance Lab Kit” (see "Sample code" paragraph above), provides a sample test method (“ApproveTaskUpdates”) to implement the Approve Task Updates test. The sample code picks a random project manager, gets the first pending task update from the list of task updates pending approval, approves it, submits the approval and finally republishes the affected project.

In order to test task update approval operations realistically, however, you also need to add a web test to hit the approvals page. You can simply set up a web test for the approvals page by using the Web Test Recorder in Visual Studio, as described above. The test mix between the web test (page hit) and the unit test described above (actual task update approval) should be set as follows (two page hits for every submission):

|  |  |  |
| --- | --- | --- |
| Operation | Test | Percentage |
| Approve Task Updates | ApproveTaskUpdatePage web test | 67 |
| ApproveTaskUpdates unit test | 33 |

### Approve Timesheets

The “EPM Stress Test” Visual Studio solution, which is part of the “Project Server 2007 Performance Lab Kit” (see "Sample code" paragraph above), provides a sample test method (ApproveTimesheets) to implement the Approve Timesheets test. The sample code picks a random timesheet approver, gets a random timesheet from the list of timesheets pending approval, and finally approves it.

In order to test timesheet approval operations realistically, however, you also need to add a web test to hit the approvals page. You can simply set up a web test for the approvals page by using the Web Test Recorder in Visual Studio, as described above. The test mix between the web test (page hit) and the unit test described above (actual timesheet approval) should be set as follows (two page hits for every submission):

|  |  |  |
| --- | --- | --- |
| Operation | Test | Percentage |
| Approve Timesheets | ApproveTimesheetPage web test | 67 |
| ApproveTimesheets unit test | 33 |

## Testing User-initiated Operations Involving Project Professional

This paragraph refers to operations involving Project Professional as the main user interface. The most frequent operations belonging to this category include the following:

* Open a project
* Save a project
* Publish a project

Since Project Professional 2007 doesn’t support multiple instances executing in the same user session, in order to test many concurrent users the use of Terminal Services is required unless you have dozens of servers. On the other end, the Visual Studio 2008 Test Agent cannot be instantiated in multiple sessions on the same machine, making it quite difficult to integrate Project Professional tests in Visual Studio Team System 2008 Test Edition. However, as already mentioned, a community-based tool capable of automating Project Professional tests inside multiple Terminal Services sessions has been developed and is available at <http://www.codeplex.com/TestFramework>. The Test Framework tool implements a controller-agent architecture conceptually similar to the Visual Studio 2008 Test Edition architecture. In a nutshell, the controller pushes information on a SQL Server database, while agents pull job information and update status information regarding their jobs from the same database. Every job is composed of one or more operations, implemented with custom .NET components and leveraging the Project Professional client object model.

The three Project Professional tests described in the following paragraphs are implemented using the Test Framework tool, while data collection is still performed with Visual Studio Team System 2008 Test Edition running at the same time.

### Open Projects

The sample code included in the Test Framework Codeplex solution shows how the opening of a project from Project Professional can be automated. The code will be dynamically loaded and run by the Test Framework agent. The Open function gets two parameters in input: the file name of the project to be opened (using ‘\*’ as the file name causes the function to automatically pick a project from the server-side list, in sequence order), and a Boolean to indicate if the project should be open in read-only or in read-write mode. The function returns the number of milliseconds elapsed for the operation, or zero in case of error. The two parameters can be set through the controller UI.

### Save Projects

Since Project Professional only sends the actual changes to the server when saving, in order to perform a meaningful save test you’ll need to apply some changes to an opened project before saving it back. Fantasy is the only limit to what can be done for generating meaningful changes before saving. One simple example is to change the start date of the project, which typically affects all of the tasks in the project, thus generating a significant change set to be saved.

The sample code included in the Test Framework Codeplex solution shows several sample functions, which can be set to be run in sequence by defining a workflow in the controller application. Two examples are:

* a ChangeStartDate function, which picks a random date in the [today ± 30 days] range and changes the project start date to the randomly selected date;
* a Save function, which simply saves back the active project; the function gets one Boolean parameter as input that can be used to skip the save operation if the active project is opened in read-only mode.

Both of the functions return the number of milliseconds elapsed for the operation, or zero in case of error.

### Publish Projects

The sample code included in the Test Framework Codeplex solution shows how to automate the publishing of a project. The Publish function gets three parameters as input: the first is a boolean indicating if the entire project (true) or only the changes (false) will be published, the second indicates the url for the optional url for the project workspace to be provisioned, and the third parameter is a boolean indicating if the publish operations needs to be skipped in case the opened project is in read-only mode. The function returns the number of milliseconds elapsed for the operation, or zero in case of error.

## Testing Background Operations

This paragraph refers to operations that are not the direct consequence of any user action, but instead are running in background as scheduled jobs. Some of the most frequent operations belonging to this category include the following:

* Cube building
* Active Directory synchronization
* Project archiving

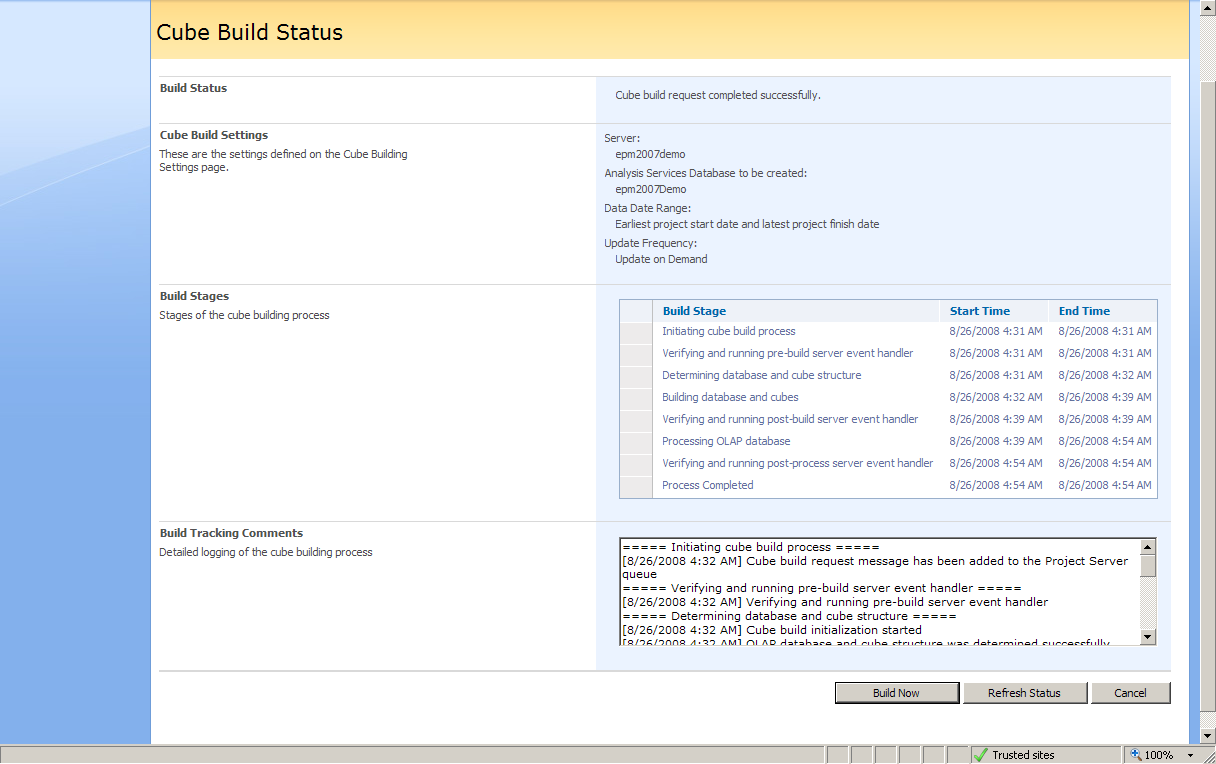
Normally, testing strategies for these operations simply consist in controlling when the operation under test will start, and running a dummy test on Visual Studio Team System 2008 Test Edition at the same time in order to capture relevant data from the affected servers, without any other load. A dummy test will need to be run in order to start the collection of performance counters data and continue for the whole duration of the cube building process. An example of a dummy test that can be used for the purpose is included in the “EPM Stress Test” Visual Studio solution (“DoNothing” class), which is part of the “Project Server 2007 Performance Lab Kit” (see "Sample code" paragraph above).

The following paragraphs provide some more specific details for testing each of the above operations.

### Cube Building

Refer to <http://technet.microsoft.com/en-us/library/cc197606.aspx> for more information on the cube building process in Microsoft Office Project Server 2007.

In order to trigger a start of the cube building process, you can simply access the Cube Build Status page under Server Settings and use the “Build Now” button at the bottom of the page.



This will cause activity on the SQL Server, the Project Server application server and the SQL Server Analysis Services server.

### Active Directory Synchronization

Refer to <http://technet.microsoft.com/en-us/library/cc197402.aspx> for more information on the Active Directory synchronization processes in Microsoft Office Project Server 2007.

The Active Directory synchronization feature in Microsoft Office Project Server 2007 can synchronize resource data in the Enterprise Resource Pool or user data in security groups. Management settings for the two types of synchronization targets are different.

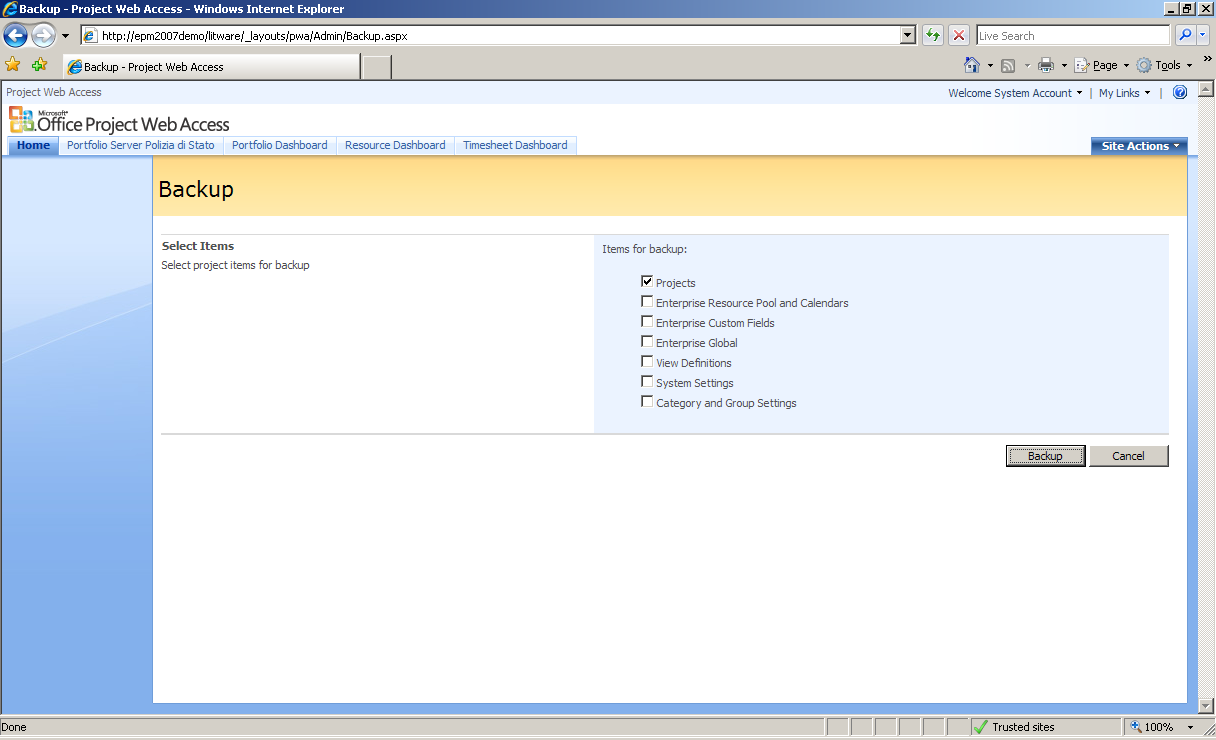
To configure and start a synchronization targeted to the Enterprise Resource Pool, you can simply access the Active Directory Enterprise Resource Pool Synchronization page from Server Settings, and use the “Save and Synchronize now” button at the bottom of the page.

To configure and start a synchronization targeted to a security group, you need two steps. First, you need to set the Active Directory group for every Project Server group you want to synchronize; you can do this by accessing the Manage Groups page from Server Settings, click on a group name to enter the Add or Edit Group page, and then click on Find Group in the Active Directory Group to Synchronize section. The second step consists in configuring the synchronization parameters and scheduling; you can do this by clicking on the Active Directory Synch Options button in the toolbar of the Manage Groups page: a dialog window will open, allowing you to set the options for group synchronization; you can use the “Save and Synchronize now” button at the bottom of this dialog window to start the synchronization task immediately.

### Project Archiving

Refer to <http://technet.microsoft.com/en-us/library/cc197603.aspx> for more information about project archiving in Microsoft Office Project Server 2007.

Project archiving (or backup) can be triggered by simply accessing the Administrative Backup page from Server Settings (select Projects and click Backup). Other entities other than projects can also be archived and tested.



The backup operation is en-queued and then processed by a Project Server application server. You can also expect significant activity on the computer running SQL Server.

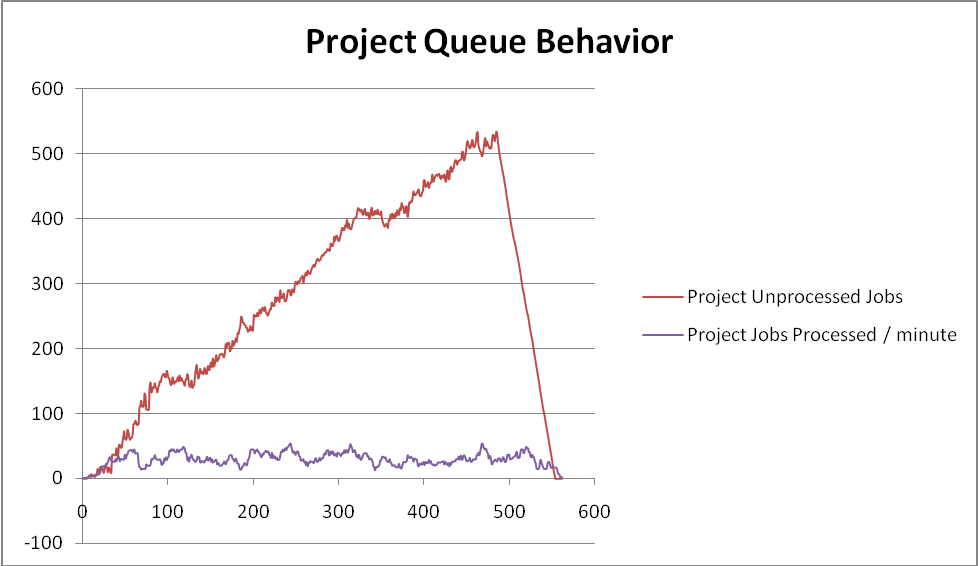
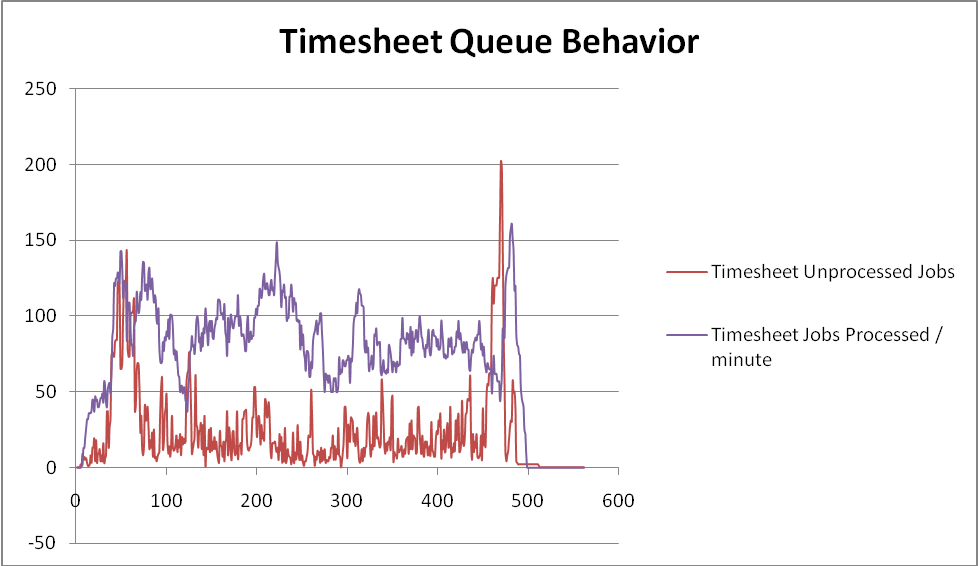
## Identify Different Queue Behaviors

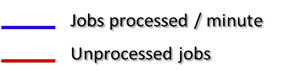
As described in the Introduction paragraph, it is important to understand and be able to identify different queue behaviors during your tests.

The two key performance counters that should be monitored for this objective are:

* QueueGeneral | Current Unprocessed Jobs (Current number of unprocessed jobs in the queue)
* QueueJobs | Jobs Processed / Minute (Number of jobs processed per minute)

The following pictures show two examples of different queue behaviors.





The picture on the left shows a situation where the load generated is not saturating the capacity of the system. The number of unprocessed jobs (red line) doesn’t show a growth trend, meaning that queued jobs can be processed at more or less the same pace as they are coming in.

The picture on the right shows instead a situation where the load generated is higher than the maximum capacity of the system at that time. The queue is piling up (red line) because the system processing throughput is lower than the incoming jobs rate. When load generation stops, the queue starts draining (drop of the red line in the right part of the graph) with the same processing throughput (blue line) until the queue is completely empty.

The different behaviors shown above can be observed in both the project and timesheet queues.

Unless you are specifically testing against a peak usage scenario, the load generation strategy for your performance tests should not cause significant queue growths and should stay in scenarios like the one depicted on the left picture above.

## Extracting and Analyzing Test Data

Other than using Visual Studio Team System 2008 Test Edition itself to open and analyze data from past test runs, you’ll also need to analyze test data with more powerful tools like Microsoft Office Excel 2007.

### Moving Data from the Lab to the Test Analysis Database

Deploying the lab environment in a isolated network is a best practice. Thus, your SQL Server database containing test data collected through Visual Studio Team System 2008 Test Edition will typically be in a separate network than the one you are connected to when performing most of the data analysis work. For this reason, it’s common to put in place a process to move test data from the lab environment to a Test Analysis environment. This can simply be done with a SQL backup / restore operation of the LoadTest database from one environment to another.

### Extracting Relevant Data from the Test Database

The first thing to know when dealing with the LoadTest database is the test run IDs for the test runs stored in the database. The GetTestRunIDs.sql file in the “Project Server 2007 Performance Lab Kit” (see "Sample code" paragraph above) contains a Transact-SQL query that can be used to get the list of test run IDs from the database, as well as other information useful to identify the test runs.

Once you have the test run ID of a test you want to analyze, you then need to extract relevant information from the database. The main table for that purpose is LoadTestComputedCounterSample, which contains all the data samples for performance counters data and key performance indicators from all the test runs. A sample Transact-SQL query which can be used to extract relevant data is included in the ExtractData.sql file in “Project Server 2007 Performance Lab Kit”. This query returns a number of result sets which can then be easily copied and pasted on an Excel worksheet for further analysis.

# Test Environment

The lab setup used for running the tests described in this white paper is depicted in the following figure.

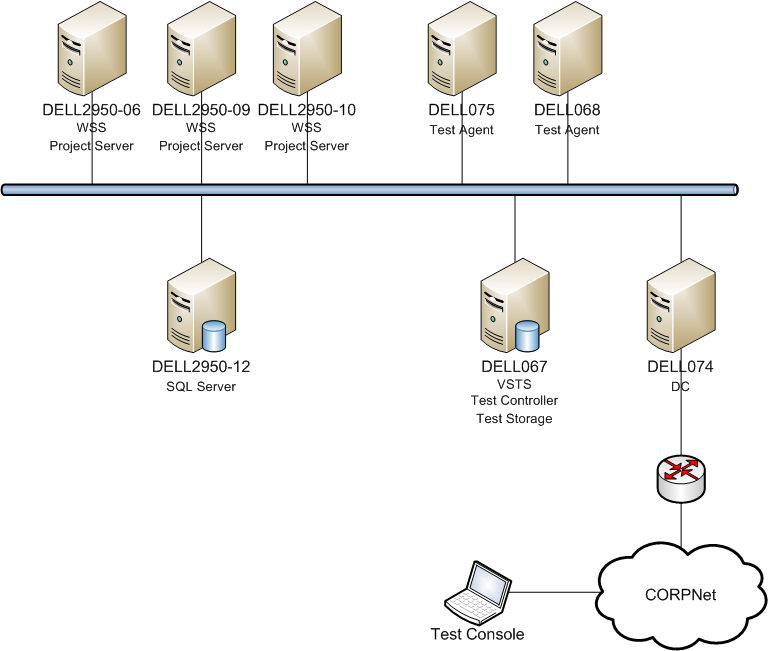


Figure 2 - Lab Setup

The three servers on the top left of the picture above have been used in several different farm topologies.

The following table shows the hardware details of all the servers involved in the lab.

| Server name | Role | Model | CPU | RAM | Storage | Network |
| --- | --- | --- | --- | --- | --- | --- |
| DELL2950-12 | SQL Server | Dell PowerEdge 2950 | 4x 1.6GHz x64 dual core  (8 cores total) | 16 GB | * Local SCSI 70 GB (C:) * 2x Local SCSI 130 GB RAID0 (E:) * 6x Fiber Channel SCSI 270 GB RAID10 (F:) * Local SCSI 210 GB (G:) * 6x Fiber Channel SCSI 270 GB RAID10 (H:) | 2x Gigabit Ethernet |
| DELL2950-06 | Project Server | Dell PowerEdge 2950 | 4x 1.6GHz x64 dual core  (8 cores total) | 16 GB | * Local SCSI 70 GB (C:) | 2x Gigabit Ethernet |
| DELL2950-09 | Project Server | Dell PowerEdge 2950 | 4x 1.6GHz x64 dual core  (8 cores total) | 16 GB | * Local SCSI 70 GB (C:) | 2x Gigabit Ethernet |
| DELL2950-10 | Project Server | Dell PowerEdge 2950 | 4x 1.6GHz x64 dual core  (8 cores total) | 16 GB | * Local SCSI 70 GB (C:) | 2x Gigabit Ethernet |
| DELL075 | Test Agent | Dell PowerEdge 2650 | 2x 2.8GHz x64 dual core (4 cores total) | 2 GB | * Local SCSI 35 GB (C:) | 2x Gigabit Ethernet |
| DELL068 | Test Agent | Dell PowerEdge 2650 | 2x 2.8GHz x64 dual core (4 cores total) | 2 GB | * Local SCSI 35 GB (C:) | 2x Gigabit Ethernet |
| DELL067 | Test Controller / Test Storage | Dell PowerEdge 2650 | 2x 2.8GHz x64 dual core (4 cores total) | 2 GB | * Local SCSI 35 GB (C:) | 2x Gigabit Ethernet |
| DELL074 | DC | Dell PowerEdge 2650 | 2x 2.8GHz x64 dual core (4 cores total) | 2 GB | * Local SCSI 35 GB (C:) | 2x Gigabit Ethernet |

For Office Project Server 2007, the patching level at the time we ran the lab was Service Pack 1 (SP1) + Infrastructure Update. We recommend that you use the latest service packs and updates to take advantage of the latest performance improvements. For information about the latest updates, see the following references:

* Description of the 2007 Microsoft Office servers Service Pack 1 and the 2007 Microsoft Office servers Language Pack Service Pack 1 (http://support.microsoft.com/kb/936984).
* Description of the Microsoft Office Servers Infrastructure Update: July 15, 2008 (<http://support.microsoft.com/kb/951297>).
* Description of 2007 Microsoft Office servers Service Pack 2 and of 2007 Microsoft Office servers Language Pack Service Pack 2 (http://support.microsoft.com/?kbid=953334).

For information about deploying Project Server 2007, see Deployment for Office Project Server 2007 on TechNet (http://technet.microsoft.com/en-us/library/cc197280.aspx).

# Test Results

Two different categories of tests have been conducted in the lab.

The first test category was designed to analyze individual loads generated on every physical architecture layer (SQL Server, Application Server, Web Front-end) by the effect of many simulated users performing a single operation concurrently.

The second category of tests is meant to simulate a mixed load of several different operations at the same time. These scenarios are for sure more realistic than the ones in the first category, but they can’t be used to measure individual impacts of specific operations on the systems. Instead, they can be used to test how the systems behave under realistic mixed loads when variables — like data profiles, server topologies, or configuration parameters like the queue settings — change.

The following paragraphs present the data profiles which were used in the lab, and the results obtained by the tests we conducted in the two categories discussed above.

## Data Profiles

The following data profiles have been generated using the EPM 2007 Test Data Population Tool for use across the different tests.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Profile | Number of users | Number of projects | Number of visible projects per user | Number of tasks per project | Number of assignments per resource |
| 1 | 1000 | 50 | 10 | 50 | 10 |
| 2 | 1000 | 200 | 10 | 50 | 10 |
| 3 | 1000 | 200 | 10 | 200 | 10 |
| 4 | 1000 | 1000 | 10 | 50 | 10 |
| 5 | 1000 | 1000 | 10 | 50 | 100 |
| 6 | 1000 | 1000 | 100 | 50 | 100 |
| 7 | 1000 | 100 | 100 | 50 | 10 |
| 8 | 1000 | 1000 | 1000 | 50 | 10 |
| 9 | 1000 | 1000 | 1000 | 100 | 100 |

Database sizes varied significantly across the different data profiles and test cycles. Total size measures for the four Project Server databases (taken after DBCC SHRINKDATABASE operations) were ranging between 42 MB (out-of-the-box situation before any data population) and about 16 GB.

## Single Operation Load Tests

Across all the single operation load tests, the farm topology was held constant: 1 front-end web server, 1 application server, 1 database server.

As already discussed above, measuring how the systems react to the load produced by a single specific operation by many simulated users at the same time can help determine a reference measure for peak periods of that specific operation. The primary goal of the single operation load tests is to measure the individual costs generated on every application role and resource class (CPU, Disk I/O, Network I/O) by the specific load under test. Derivation of main cost values from performance data and hardware configuration data is described in the following table.

|  |  |  |
| --- | --- | --- |
| Device | Operation Cost | Derivation |
| CPU | MCycle count | (% Processor Time Total) \* (processor count \* MHz per processor) / (operations/sec) |
| Disk | Read size (bytes) | (Physical Disk Reads Bytes/sec [*Appropriate drive letters*]) /  (operations /sec) |
| Write size (bytes) | (Physical Disk Writes Bytes/sec [*Appropriate drive letters*]) /  (operations /sec) |
| Network | Send size (bytes) | (Network Interface Bytes Sent/sec [*Appropriate NIC instances*] /  (operations /sec) |

### Access PWA Home Page

#### Input parameters

By using all the different data profiles (see "Data Profiles" above), the following parameters were varied across different test runs:

* Number of projects
* Number of visible projects per user
* Number of tasks per project
* Number of assignments per user

The only parameter that influenced measured data significantly was the number of visible projects per user. This was expected because the default PWA home page contains a list with all the projects the user can access.

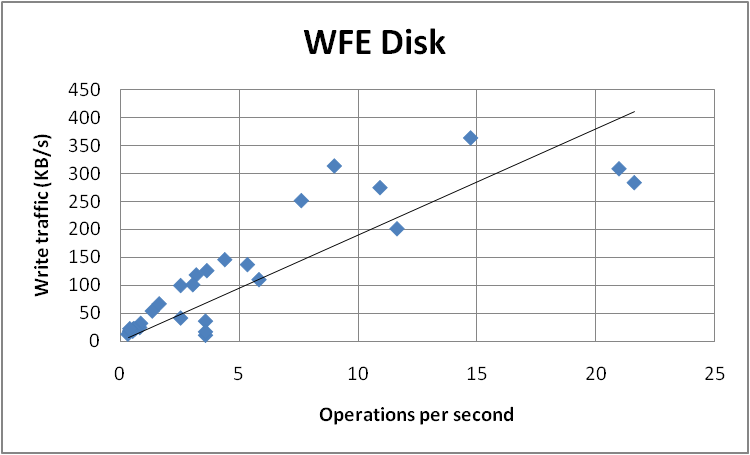
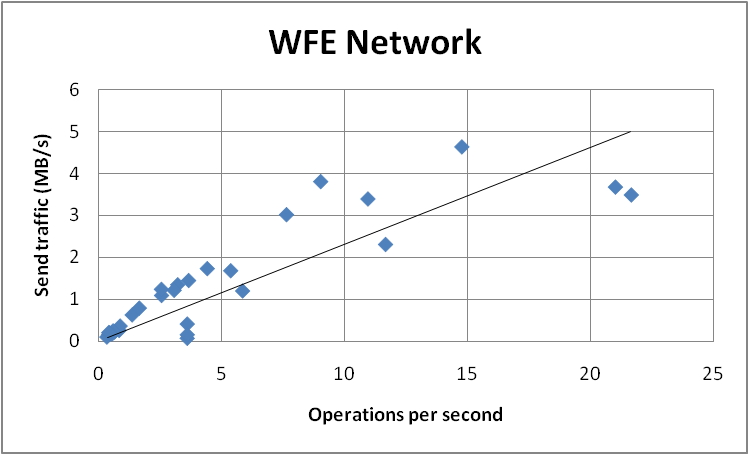
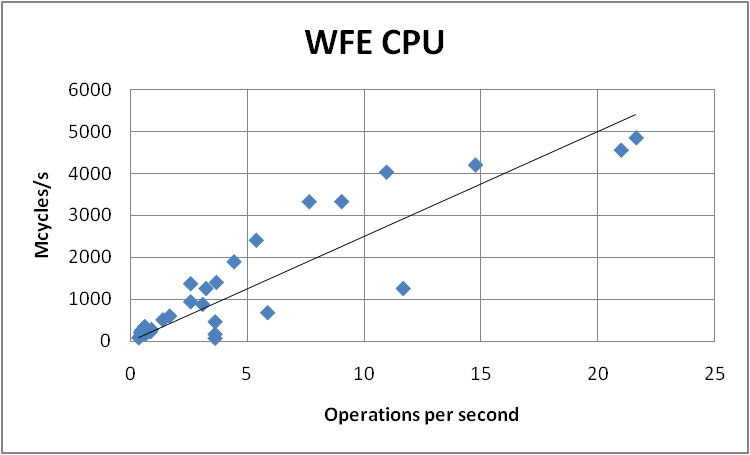
Moreover, the number of concurrent users was varied (30 to 150), leading to different throughputs obtained across the tests.

#### Results

In terms of throughput, we observed average results ranging from 2.5 to 21.6 operations per second, with peaks above 40.

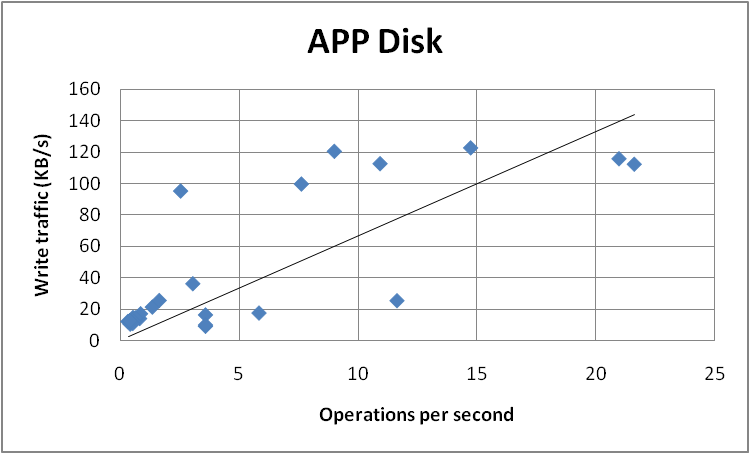
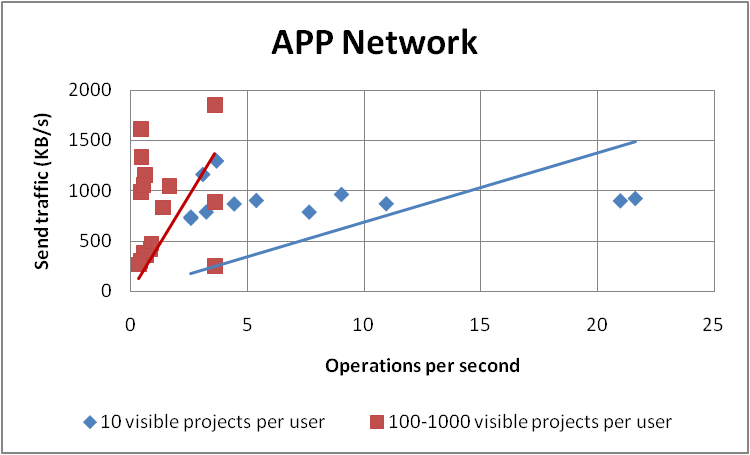
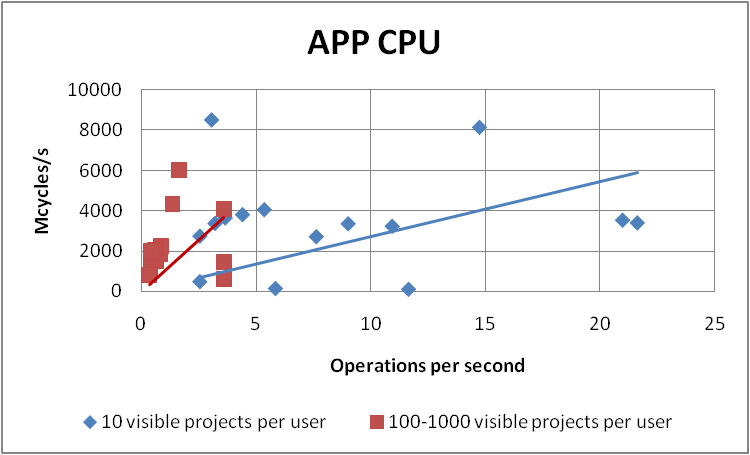
The following charts show the measured operation costs, and the estimated best-fit lines, as functions of throughput.

##### Web Front-End Server



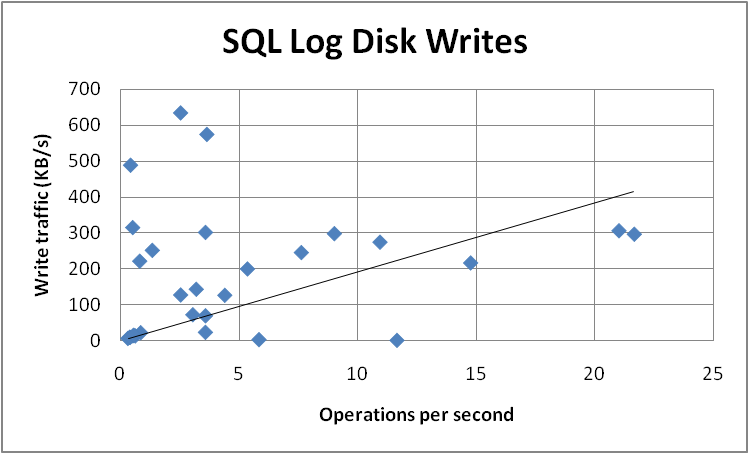
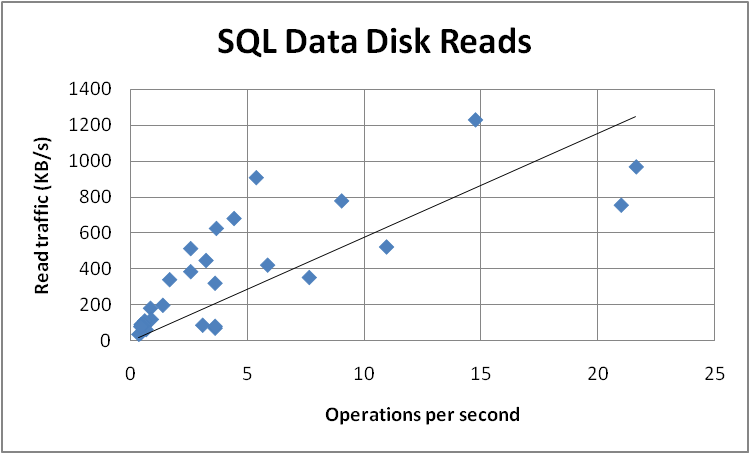
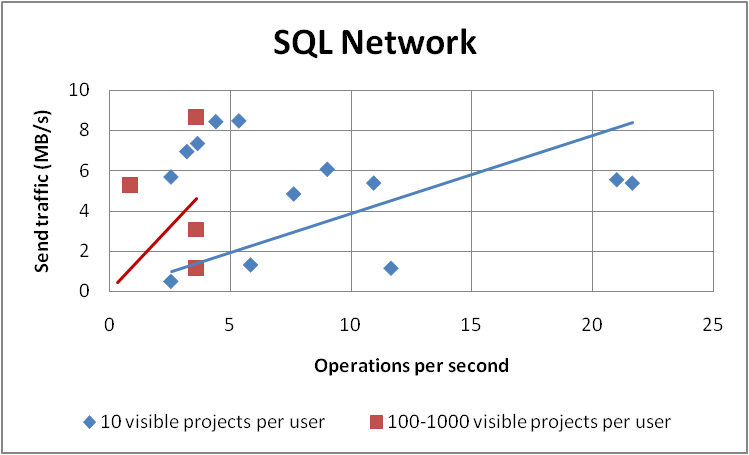
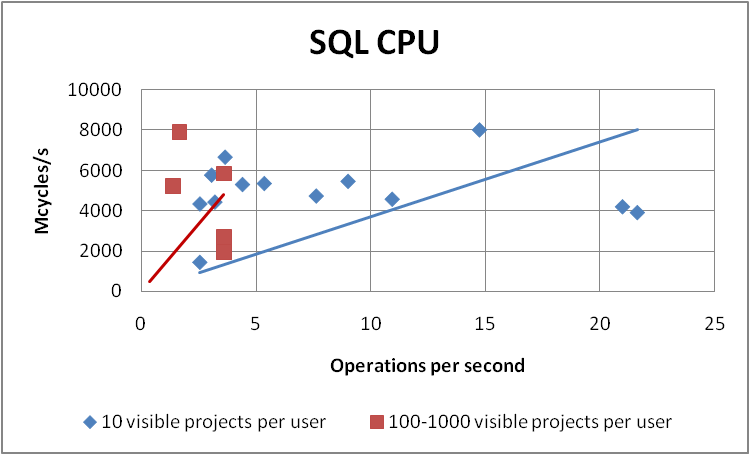
Read disk operations in the Web Front-End role are negligible, while write operations can be referred mainly to log writes.

##### Application Server



Read disk operations in the Application Server role are negligible, while write operations can be referred mainly to log writes.

##### SQL Server



### Access Project Center Views

#### Input parameters

By using all the different data profiles (see "Data Profiles" above), the following parameters were varied across different test runs:

* Number of projects
* Number of visible projects per user
* Number of tasks per project

The only parameter that influenced measured data significantly was the number of visible projects per user. This was expected because the Project Center views contain all the projects the user can see.

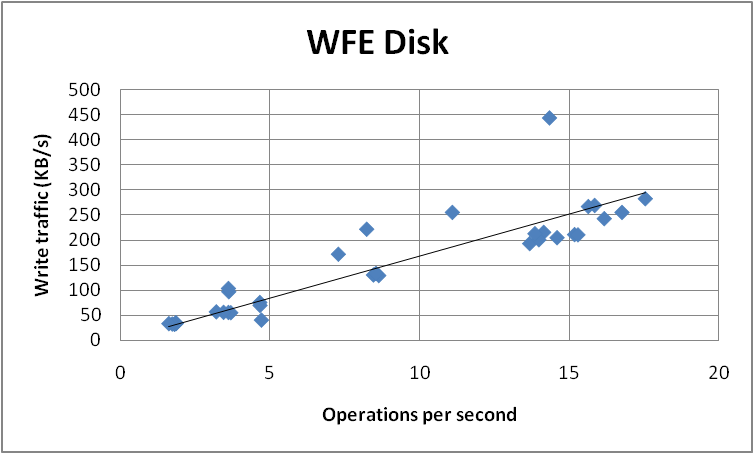
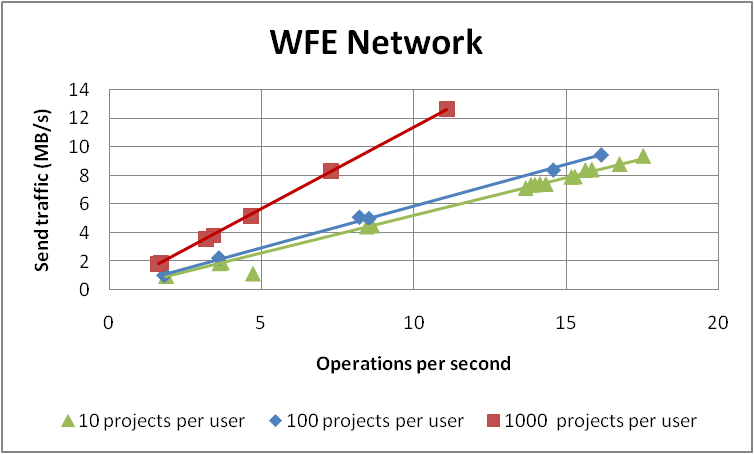
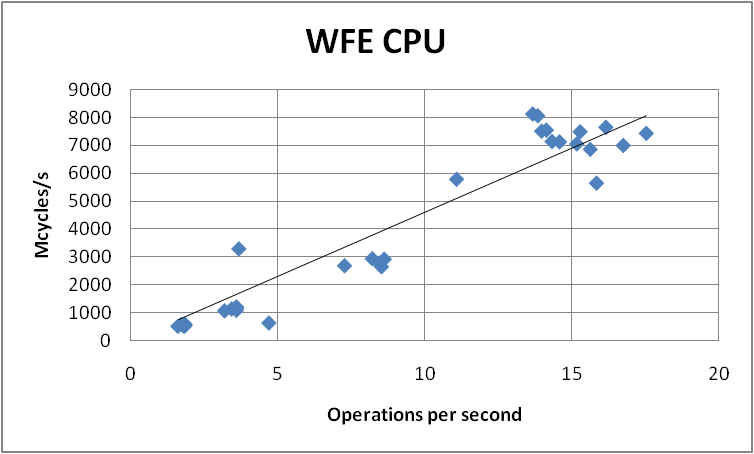
Moreover, the number of concurrent users was varied (10 to 90), leading to different throughputs obtained across the tests.

#### Results

In terms of throughput, we observed average results ranging from 1.6 to 17.5 operations per second.

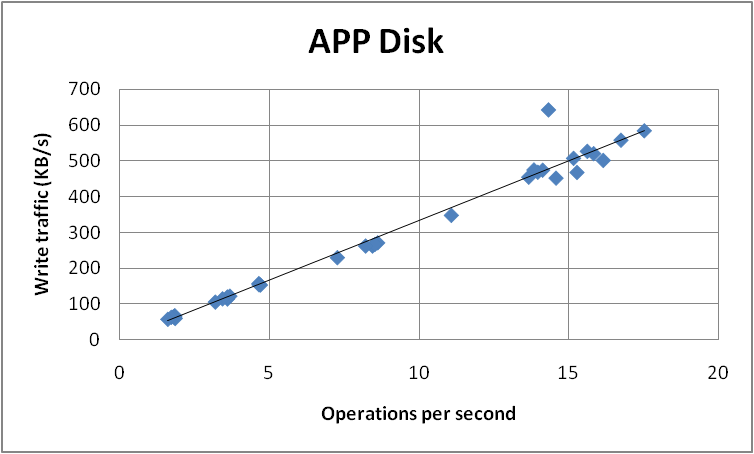
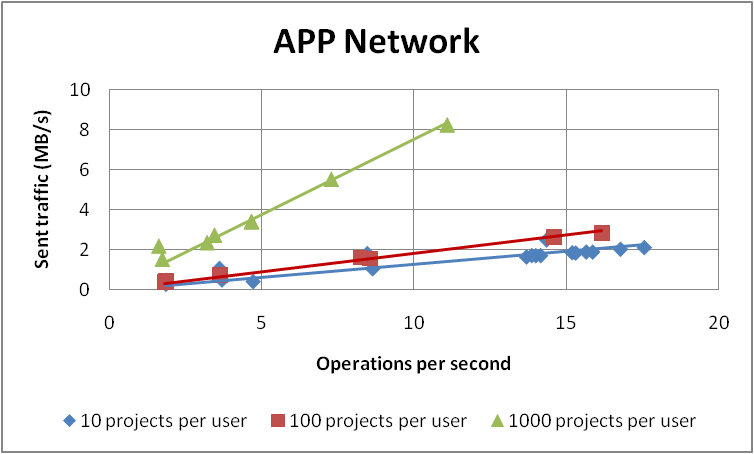
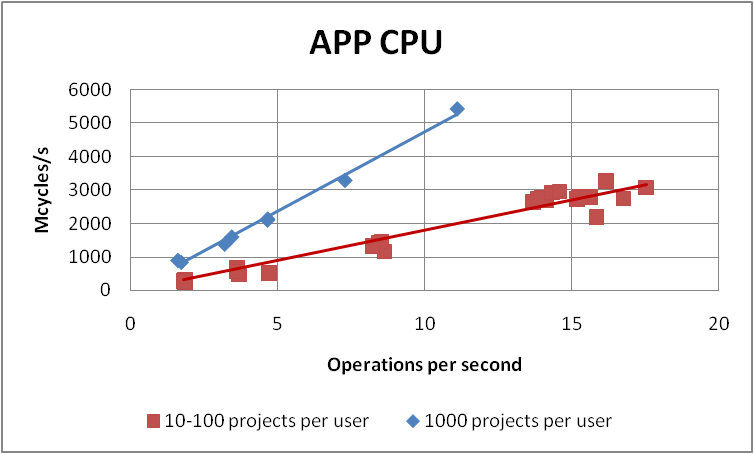
The following charts show the measured operation costs, and the estimated best-fit lines, as functions of throughput.

##### Web Front-End Server



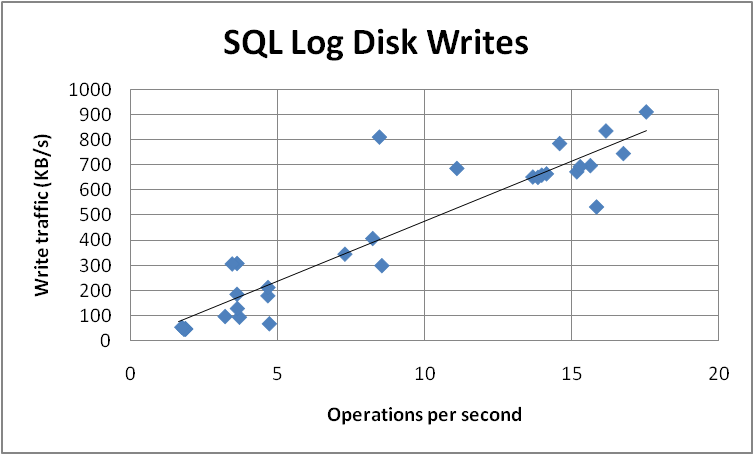
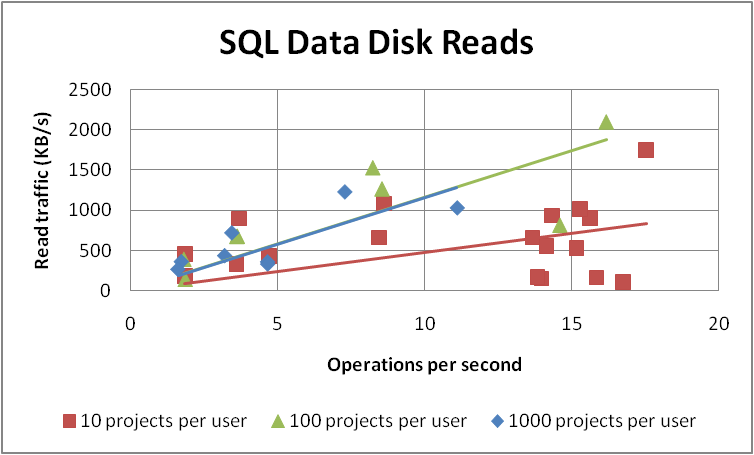
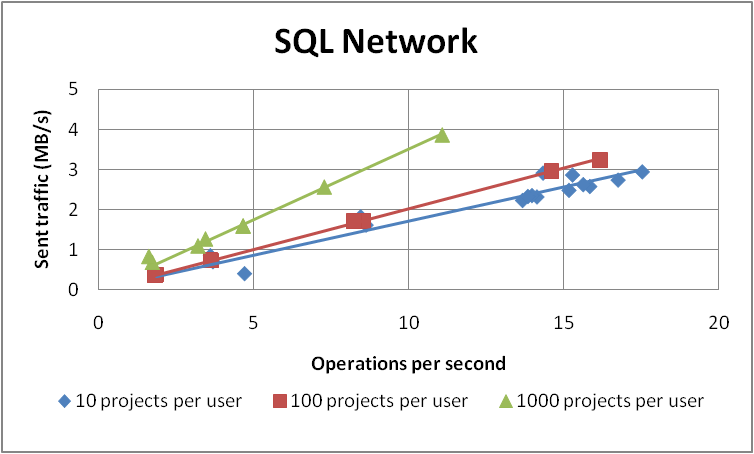
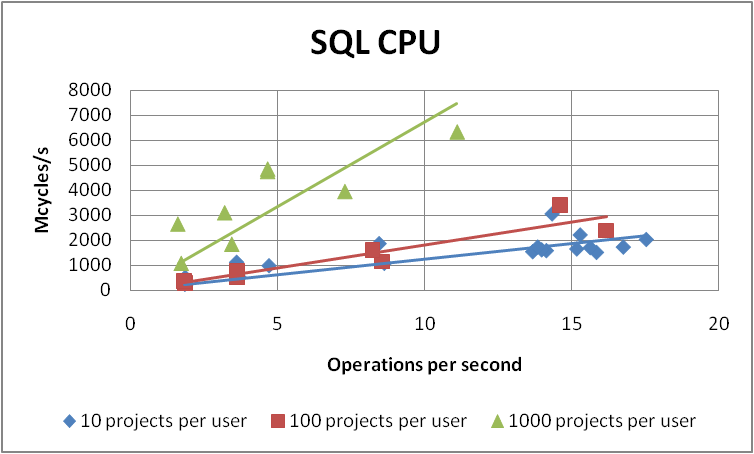
Read disk operations in the Web Front-End role are negligible, while write operations can be referred mainly to log writes.

##### Application Server



Read disk operations in the Application Server role are negligible, while write operations can be referred mainly to log writes.

##### SQL Server



### Access Resource Center Views

#### Input parameters

The following parameters were varied across different test runs:

* Number of visible resources per user

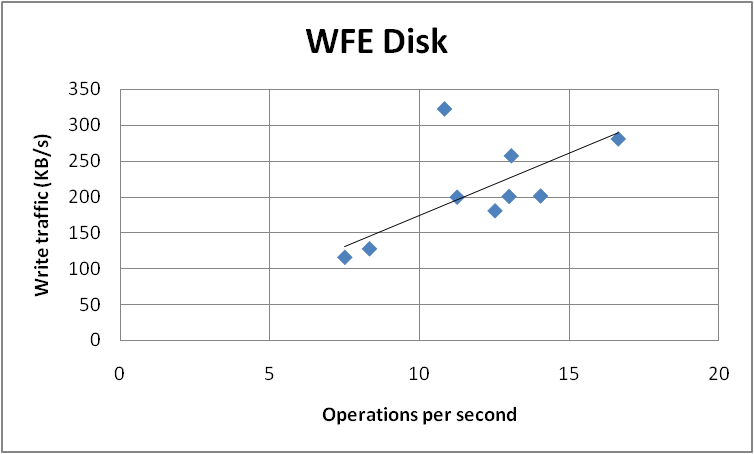
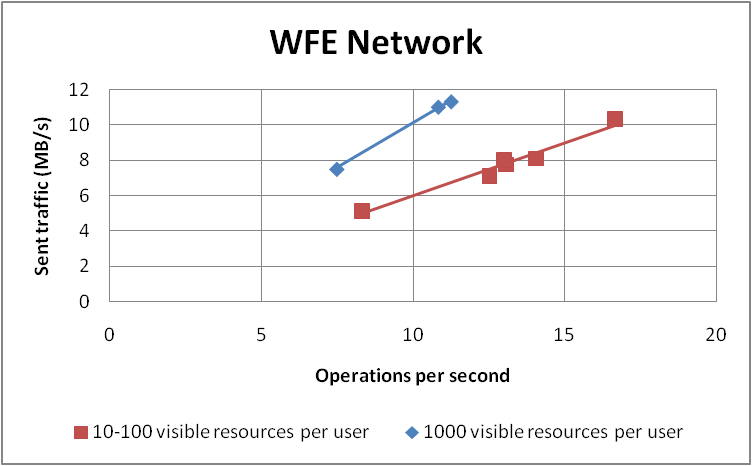
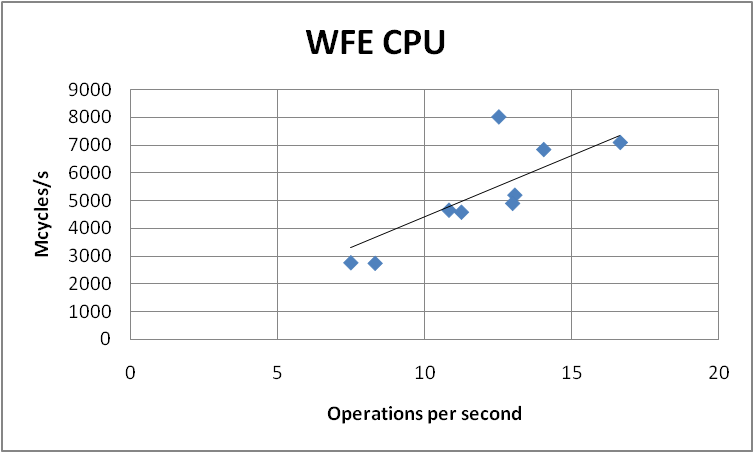
Moreover, the number of concurrent users was varied (30 to 80), leading to different throughputs obtained across the tests.

#### Results

In terms of throughput, we observed average results ranging from 7.5 to 16.6 operations per second.

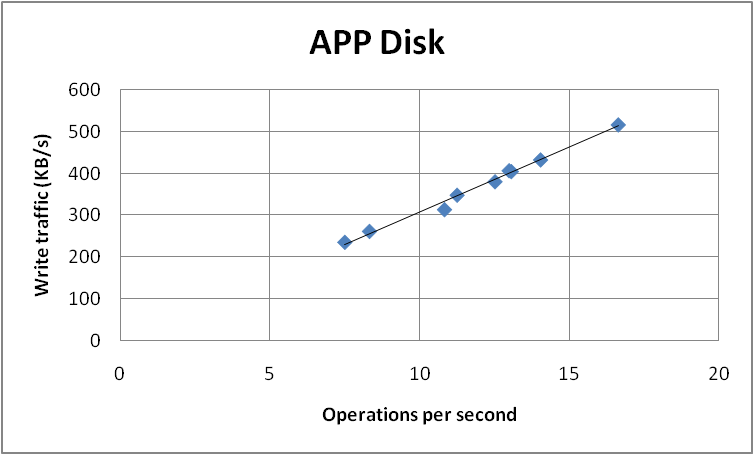
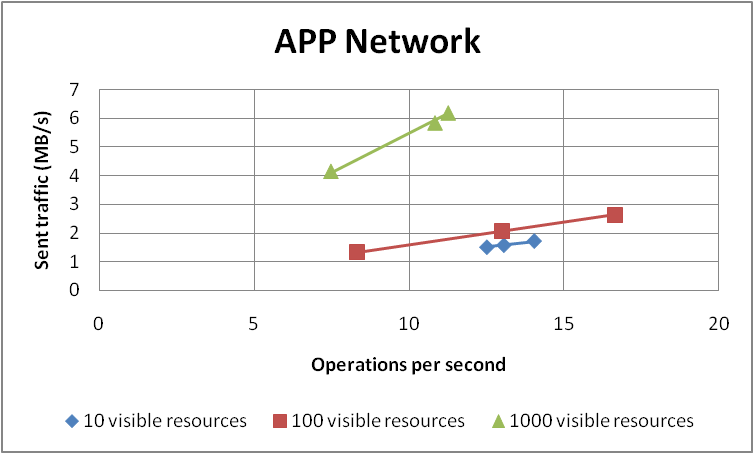
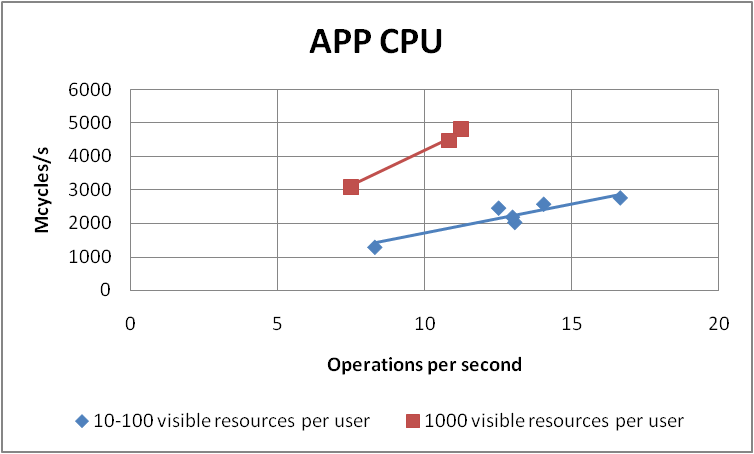
The following charts show the measured operation costs, and the estimated best-fit lines, as functions of throughput.

##### Web Front-End Server



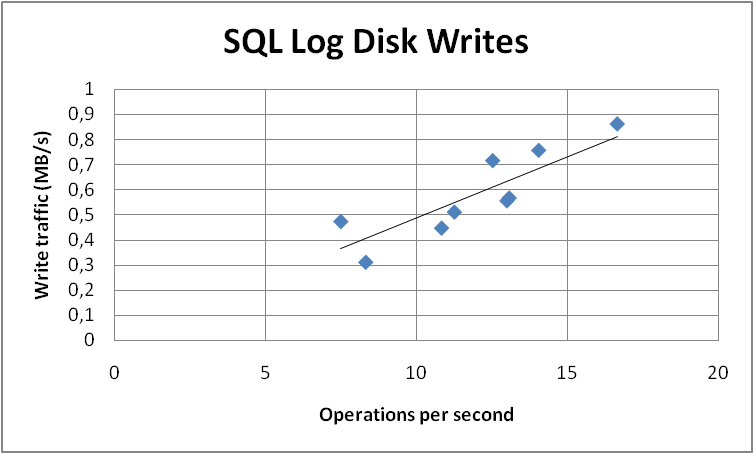
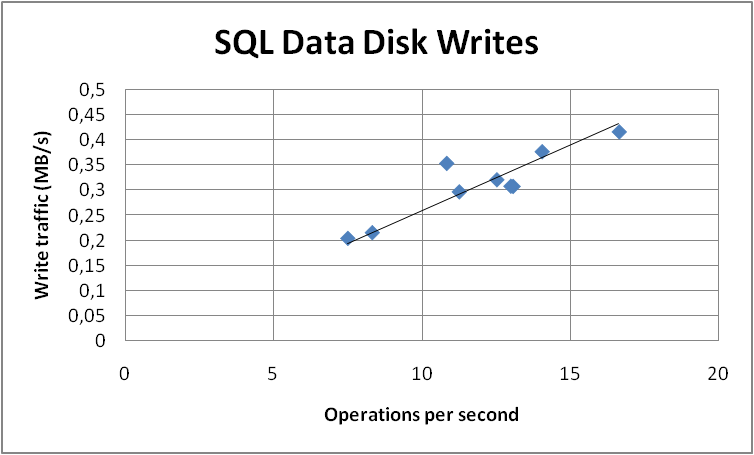
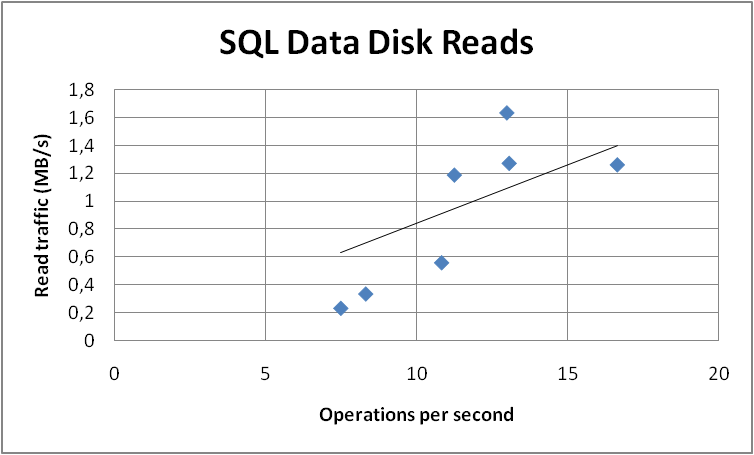
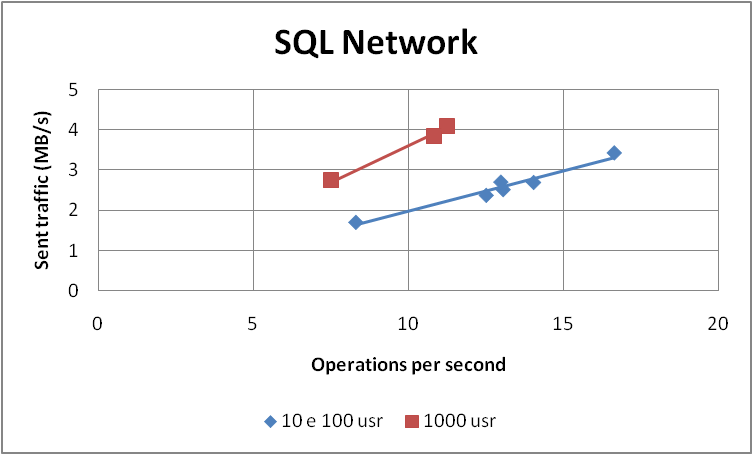
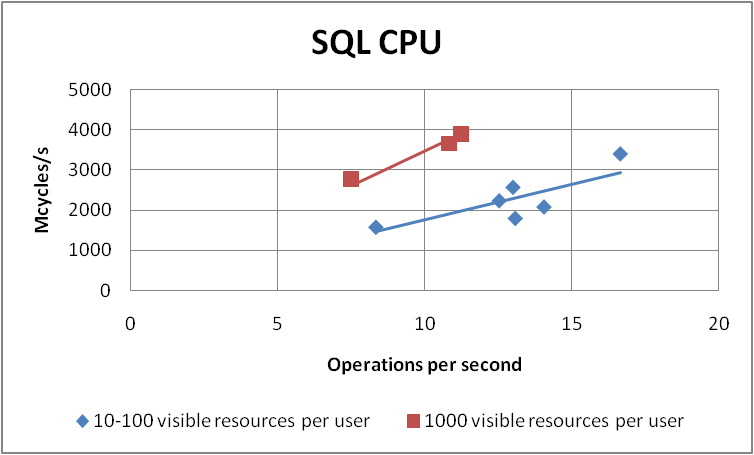
Read disk operations in the Web Front-End role are negligible, while write operations can be referred mainly to log writes.

##### Application Server



Read disk operations in the Application Server role are negligible, while write operations can be referred mainly to log writes.

##### SQL Server



### Submit Task Updates

#### Input parameters

By using data profiles 2, 5, and 9 (see "Data Profiles" above), the following parameters were varied across different test runs:

* Number of projects
* Number of visible projects per user
* Number of tasks per project
* Number of assignments per resource

The only parameter that influenced measured data significantly was the number of assignments per resource. This was expected because the My Tasks page contains all the assignments for the current user.

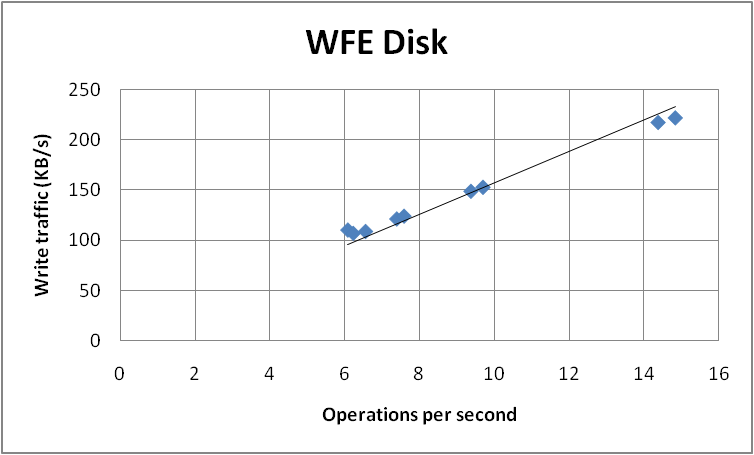
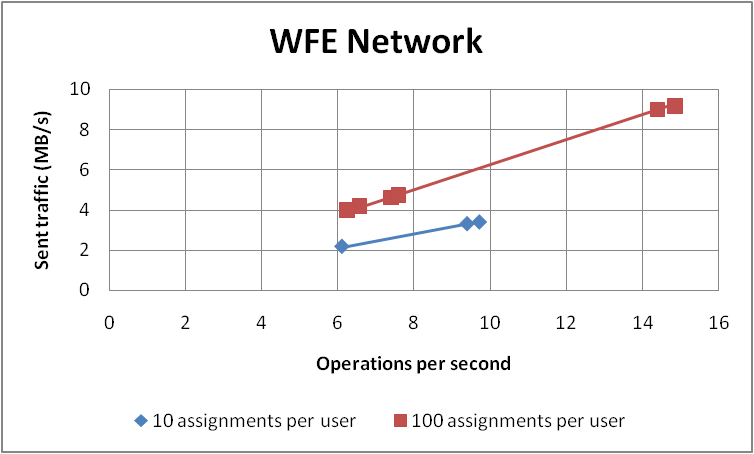
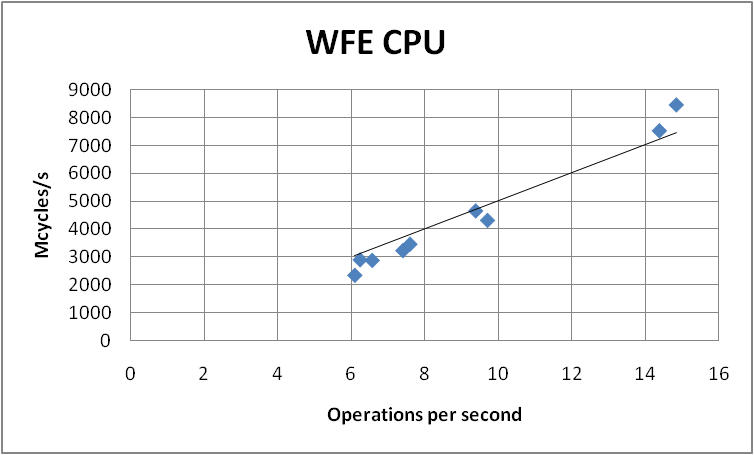
Moreover, the number of concurrent users was varied (30 to 40), leading to different throughputs obtained across the tests.

#### Results

In terms of throughput, we observed average results ranging from 6 to 14.9 operations per second.

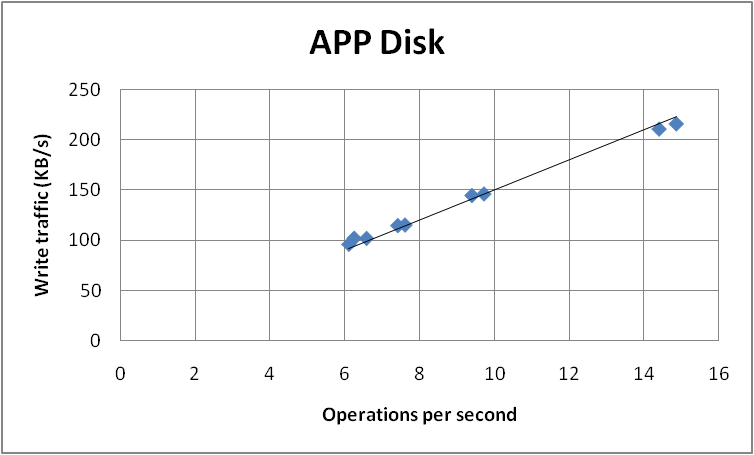
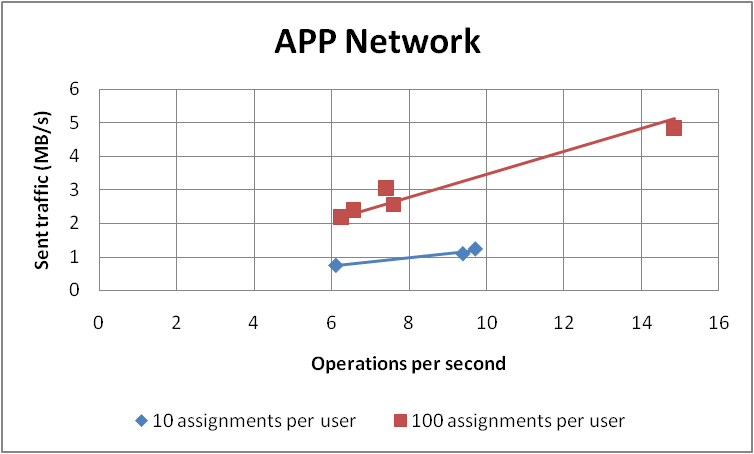
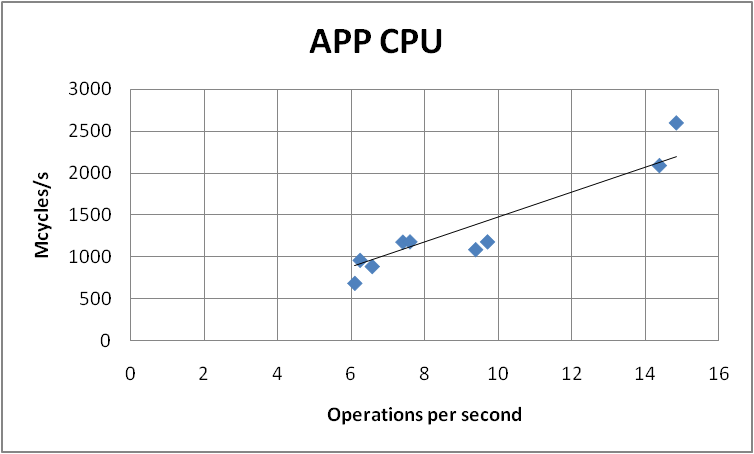
The following charts show the measured operation costs, and the estimated best-fit lines, as functions of throughput.

##### Web Front-End Server



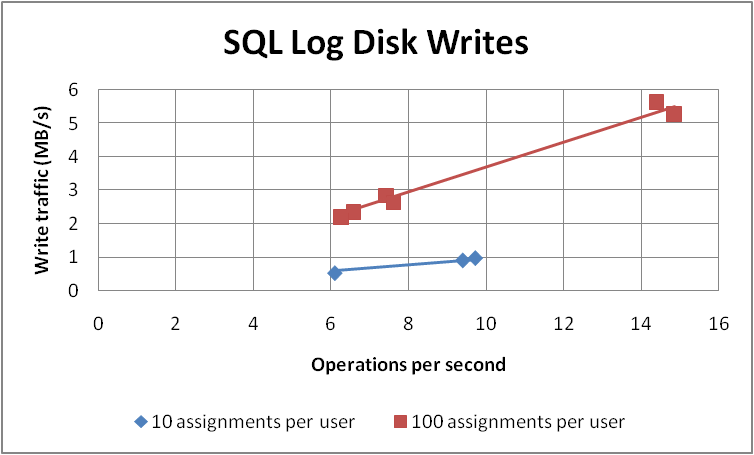
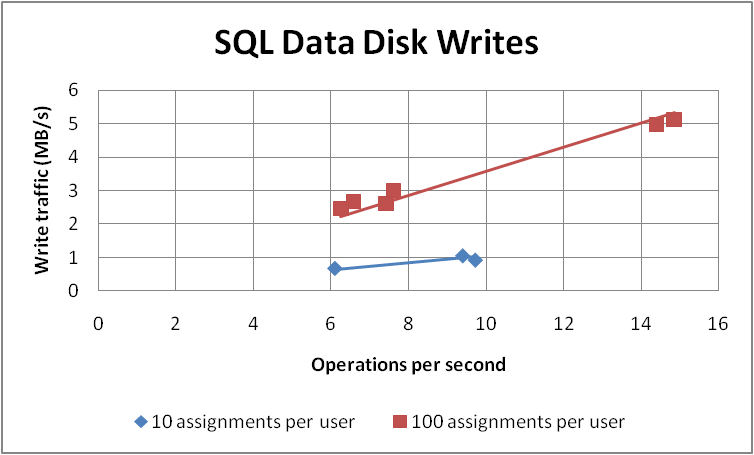
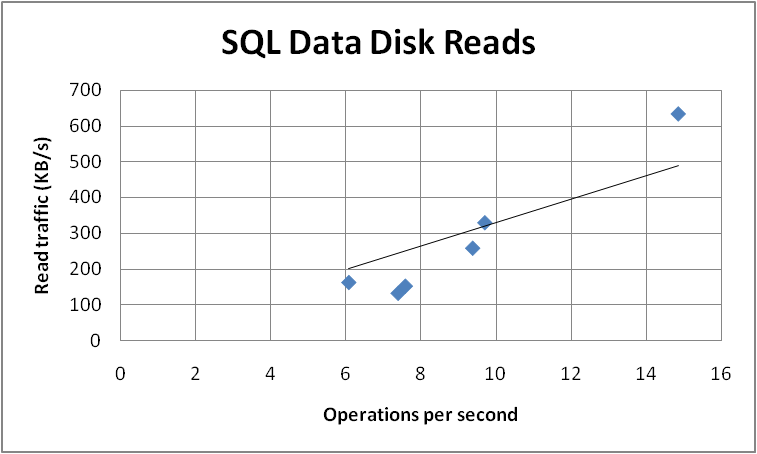
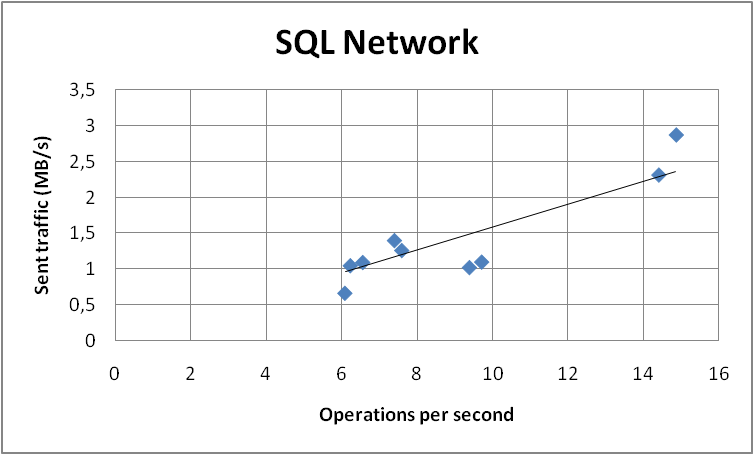
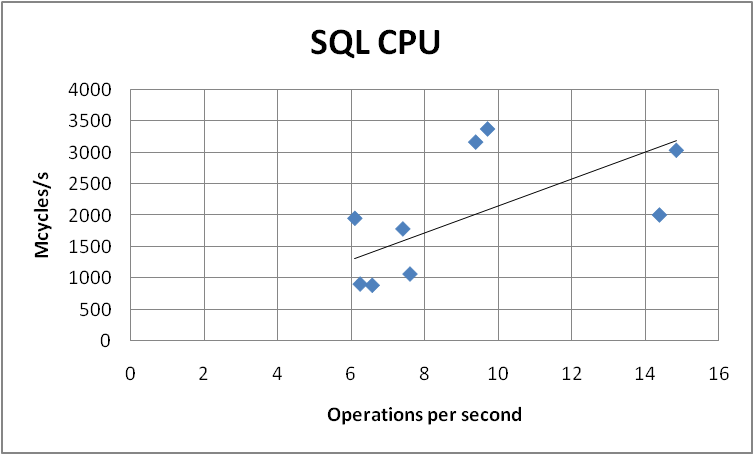
Read disk operations in the Web Front-End role are negligible, while write operations can be referred mainly to log writes.

##### Application Server



Read disk operations in the Application Server role are negligible, while write operations can be referred mainly to log writes.

##### SQL Server



### Submit Timesheets

This is a write operation that is managed through the Timesheet queue in Project Server. For this reason, we split the test in two parts: the first one is the user interface and en-queue operation, the second one is the de-queue operation. In order to separate the two parts, we stopped the queue service on the application server while submitting timesheets, and for the second part we restarted the queue service without submitting any other timesheet.

#### Input parameters

By using data profiles 1, 2, 5, and 9 (see "Data Profiles" above), the following parameters were varied across different test runs:

* Number of projects
* Number of visible projects per user
* Number of tasks per project
* Number of assignments per resource

The only parameter that influenced measured data significantly was the number of assignments per resource. This was expected because the timesheets were pre-populated with all the assignments for the current user.

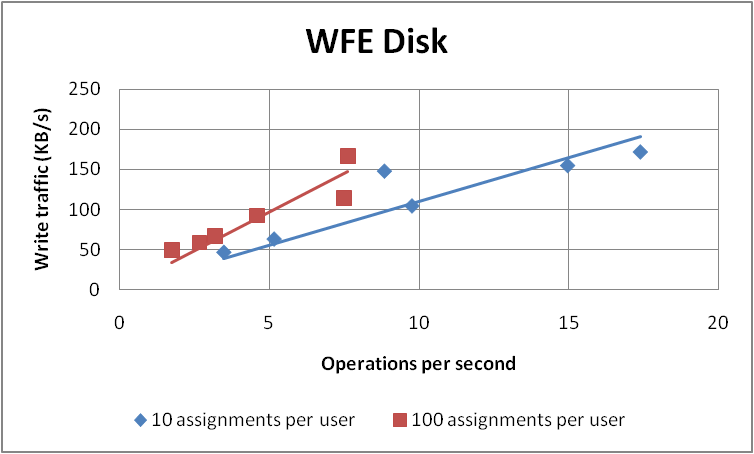
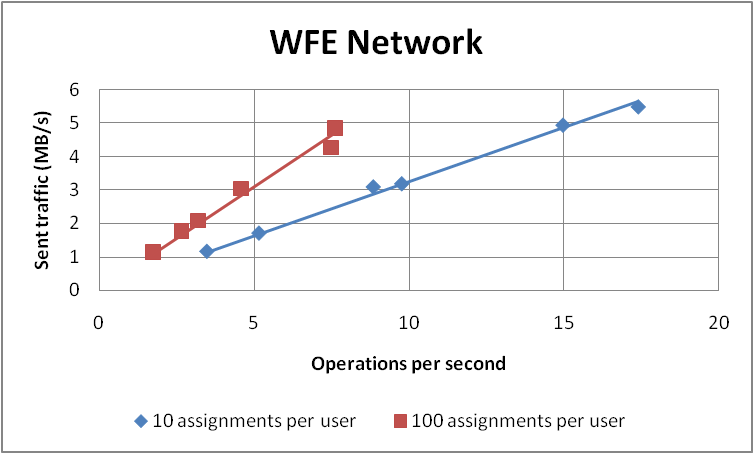
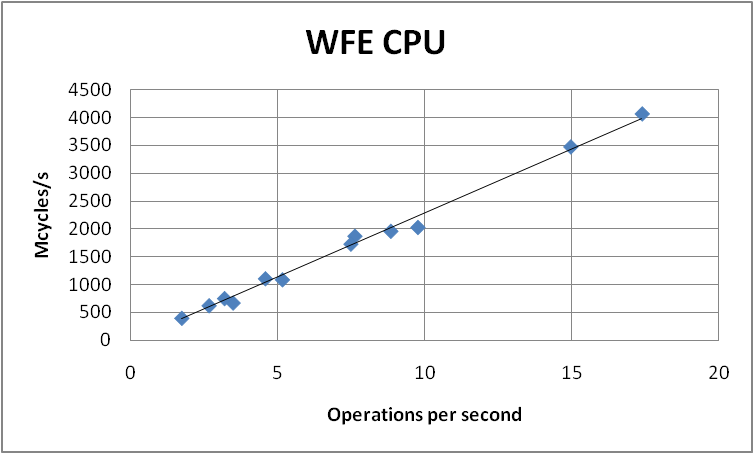
Moreover, the number of concurrent users was varied (20 to 40), leading to different throughputs obtained across the tests.

#### Results

In terms of throughput, we observed average results ranging from 1.7 to 17.4 operations per second for the user interface and en-queue operations. The de-queue operation throughputs ranged from 1 to 9.6 operations per second.

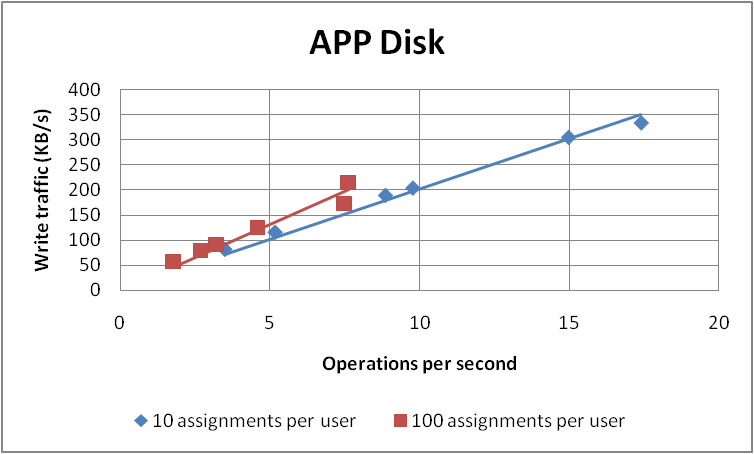
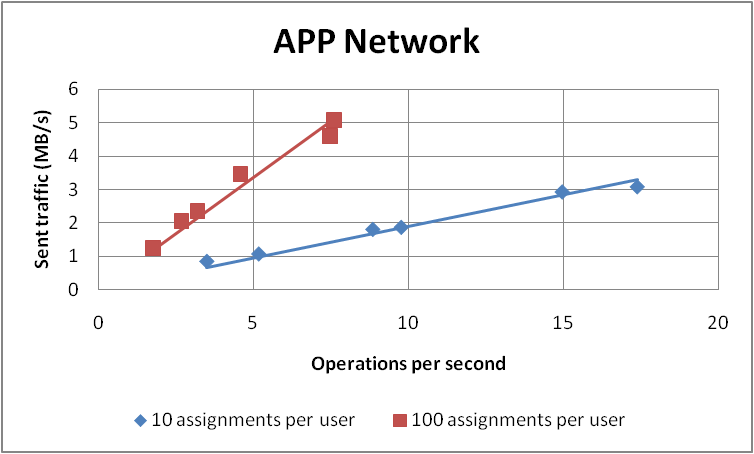
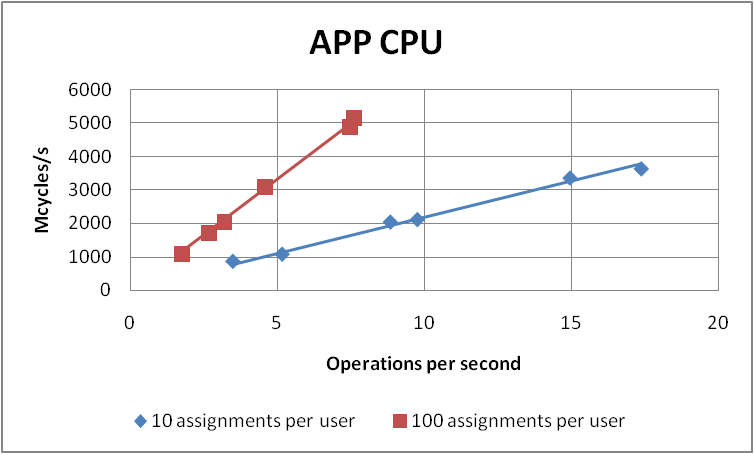
The following charts show the measured operation costs, and the estimated best-fit lines, as functions of throughput.

##### Web Front-End Server – User interface + en-queue operations

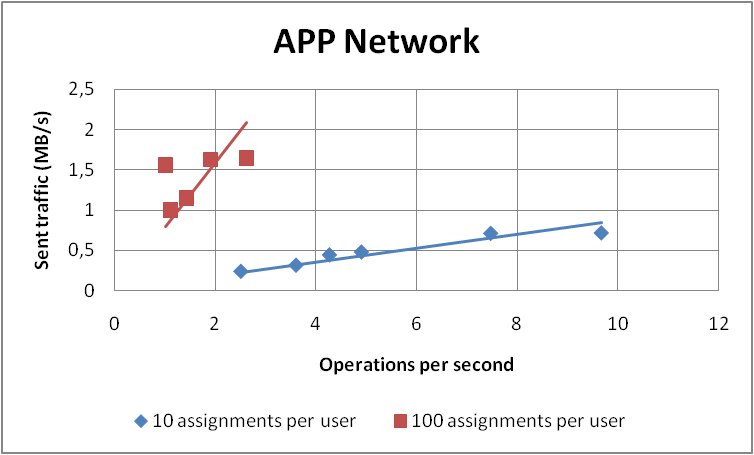
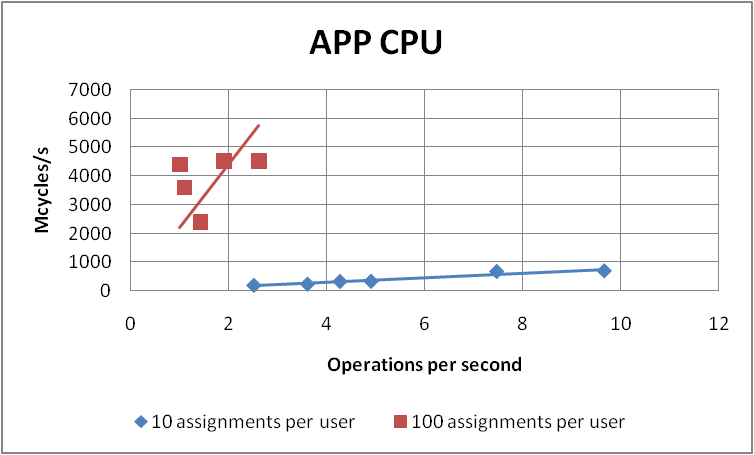


Read disk operations in the Web Front-End role are negligible, while write operations can be referred mainly to log writes.

##### Application Server – User interface + en-queue operations

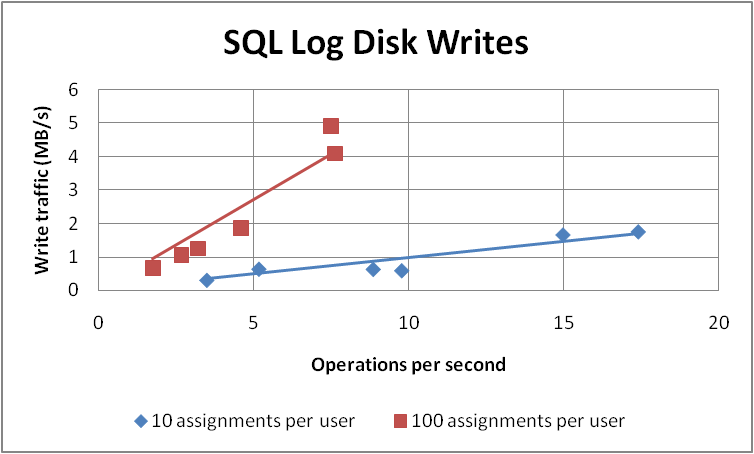
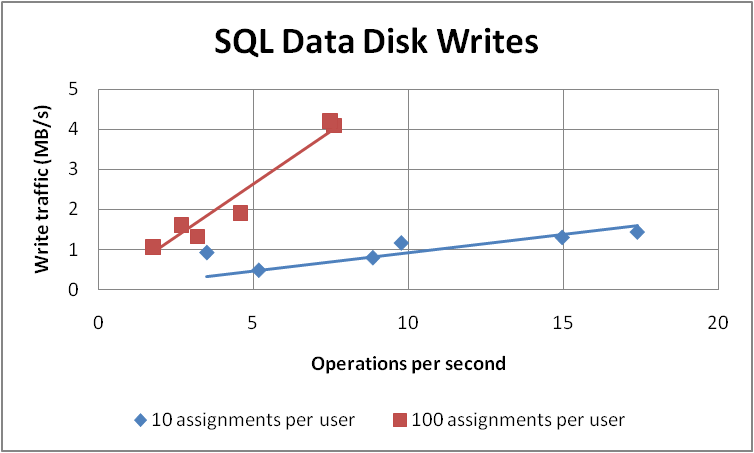
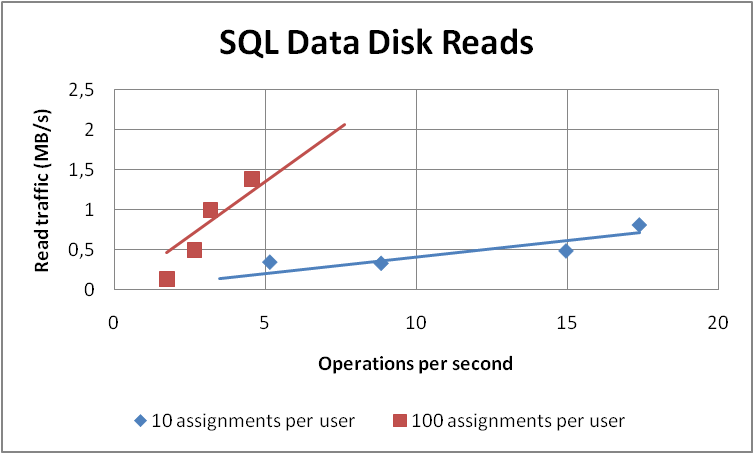
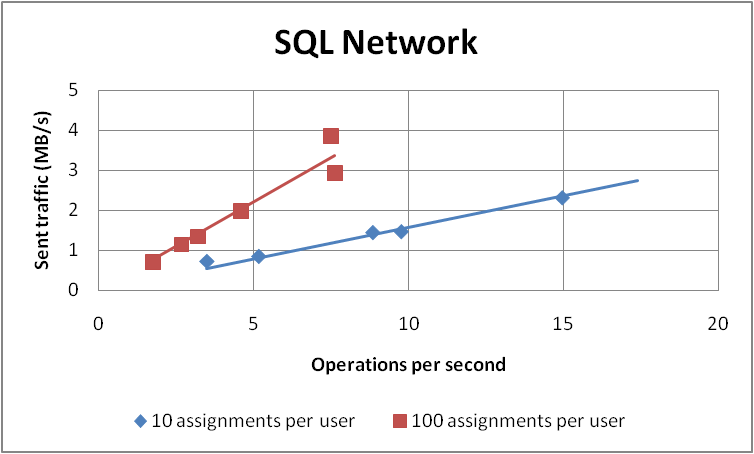
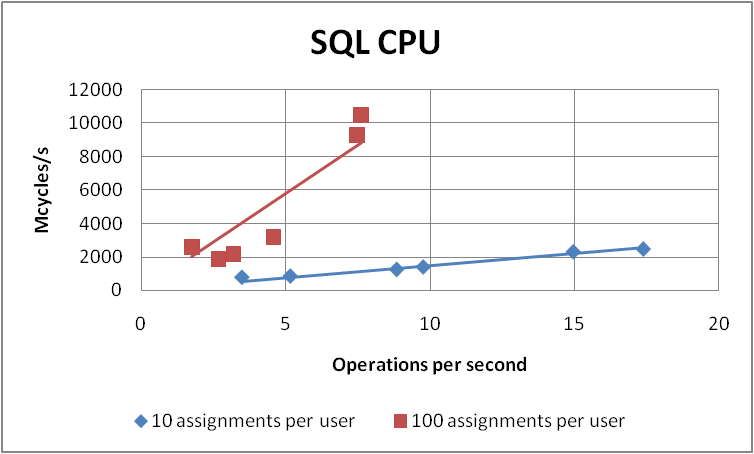


##### Application Server – De-queue operations

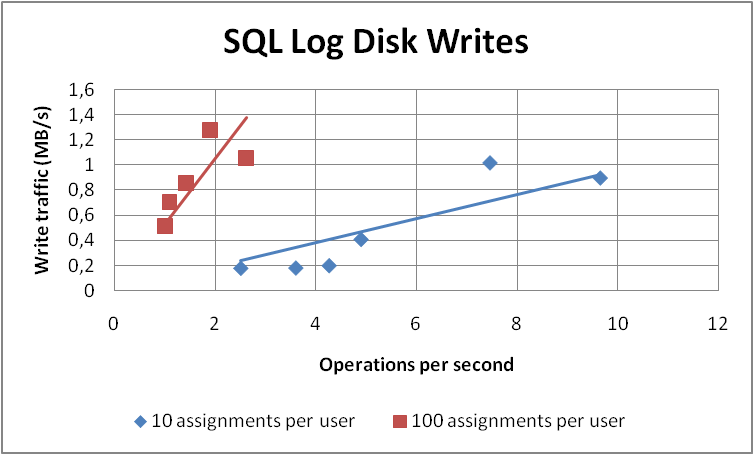
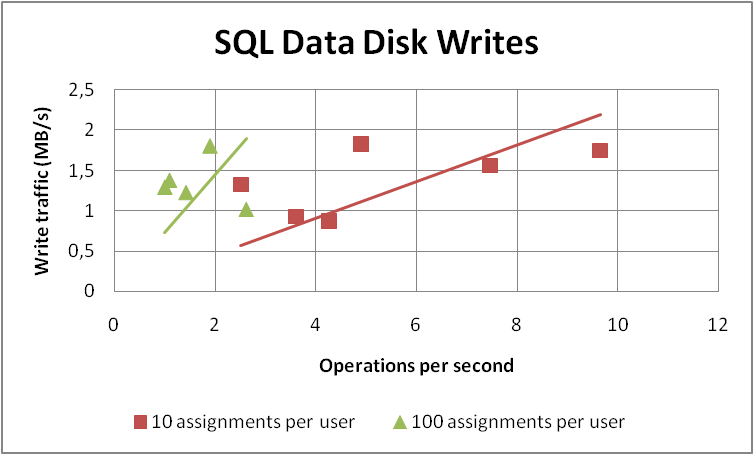
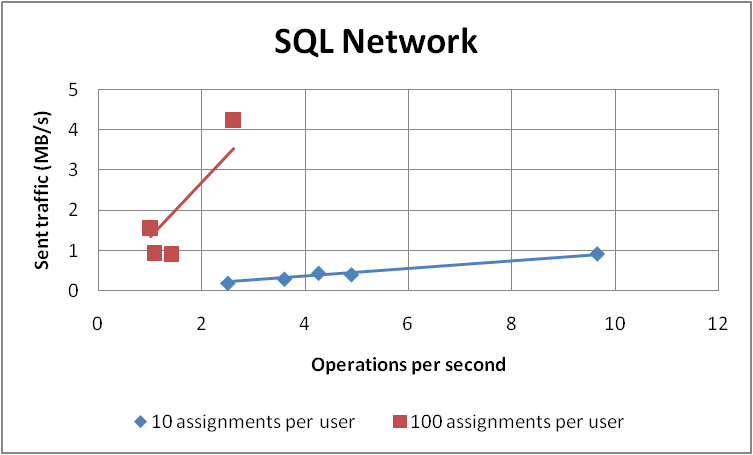
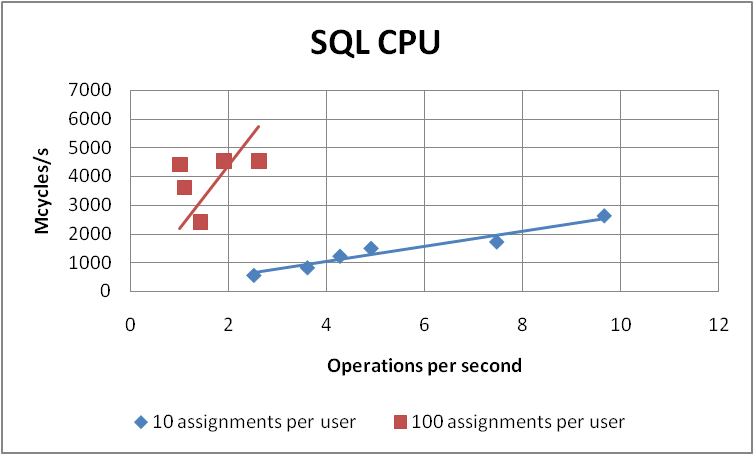


In both cases, read disk operations in the Application Server role are negligible. Write operations can be referred mainly to log writes for user interface and en-queue operations, and are negligible for the de-queue operations.

##### SQL Server – User interface + en-queue operations



##### SQL Server – De-queue operations



For de-queue operations, SQL data disk reads are negligible.

### Approve Task Updates

#### Input parameters

By using data profiles 2, 5, and 9 (see "Data Profiles" above), the following parameters were varied across different test runs:

* Number of projects
* Number of visible projects per user
* Number of tasks per project
* Number of assignments per resource

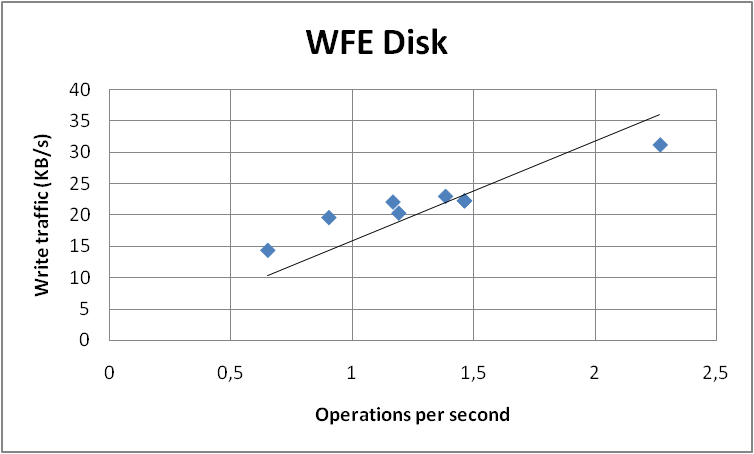
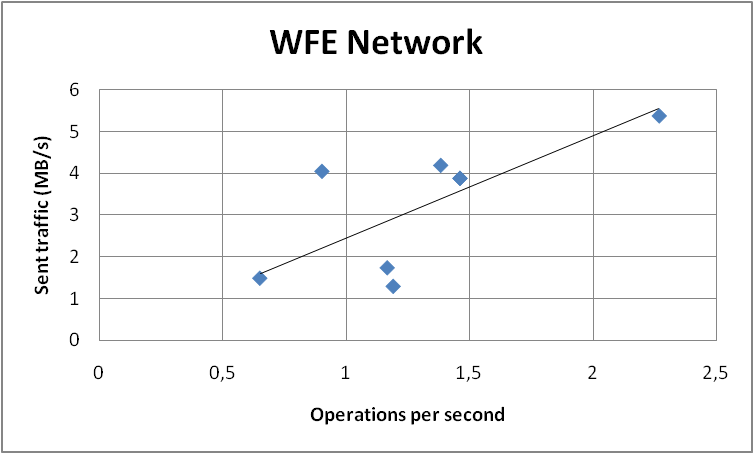
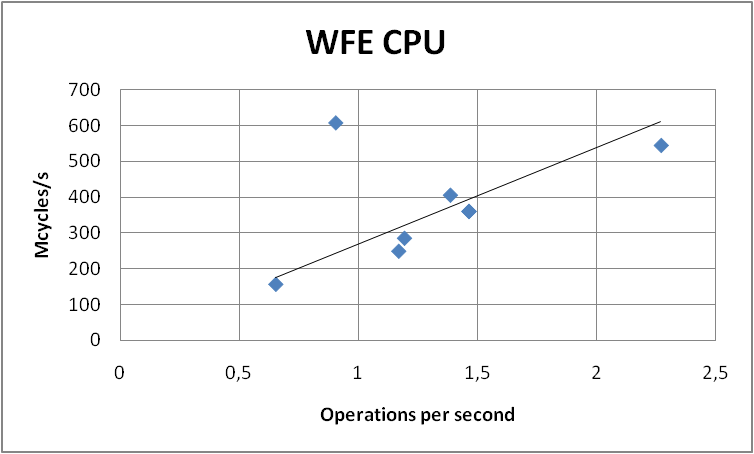
None of the parameters above influenced measured data significantly. This was expected because the task updates operations were referring to one task at a time.

#### Results

In terms of throughput, we observed average results ranging from 0.6 to 2.3 operations per second.

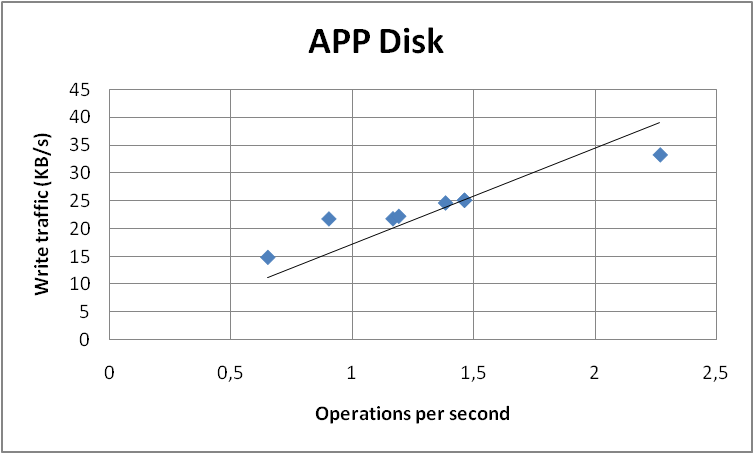
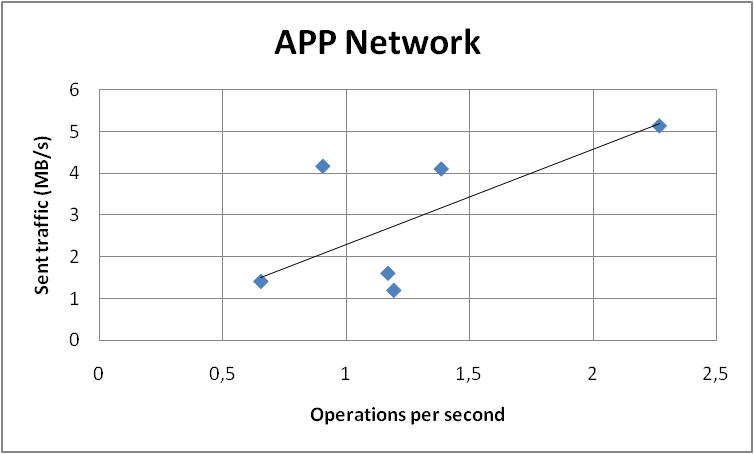
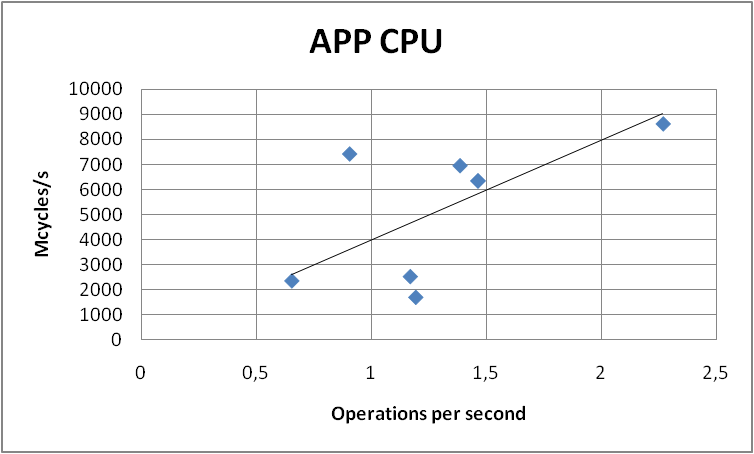
The following charts show the measured operation costs, and the estimated best-fit lines, as functions of throughput.

##### Web Front-End Server



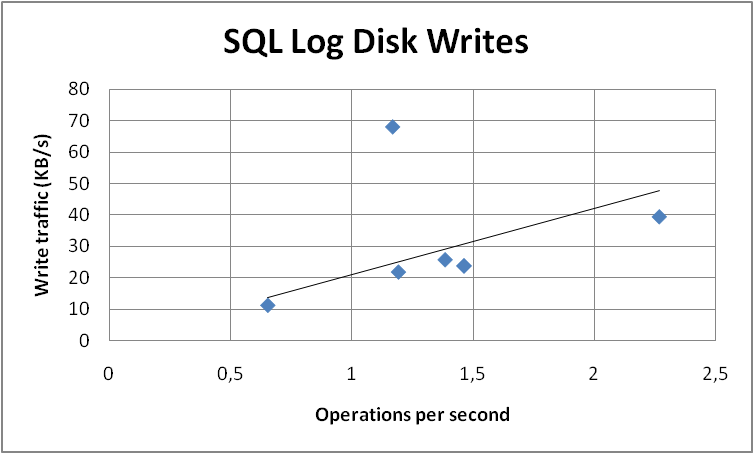
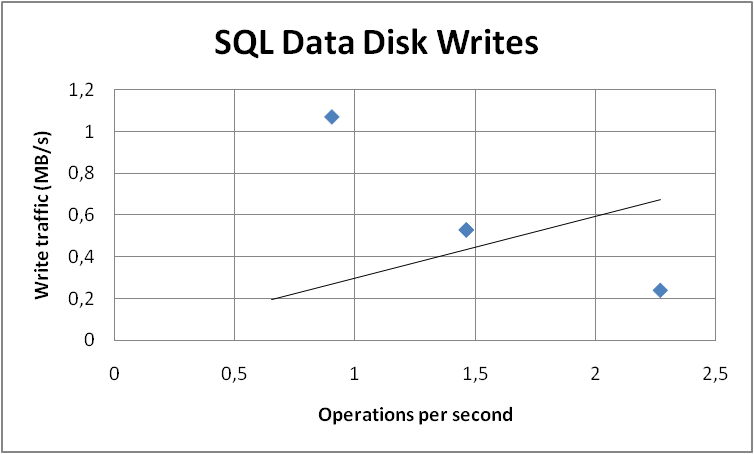
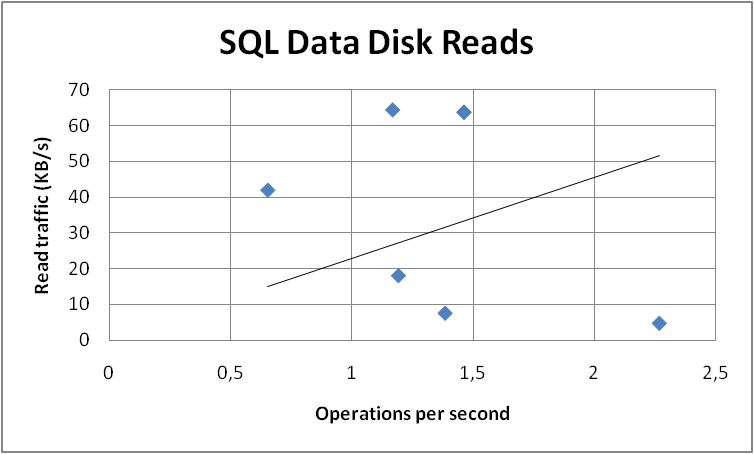
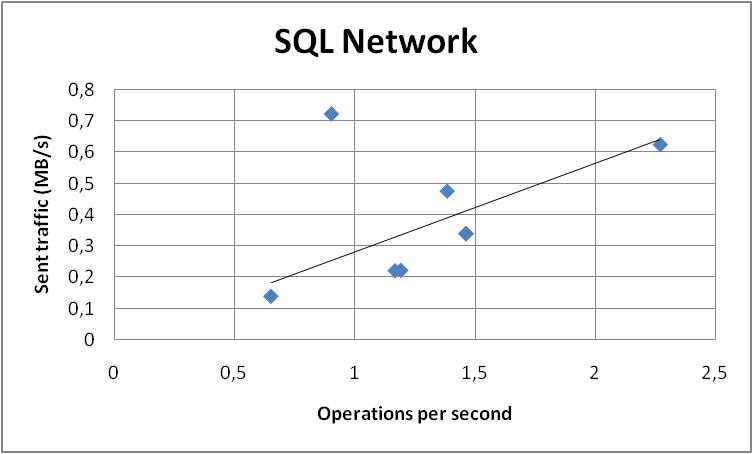
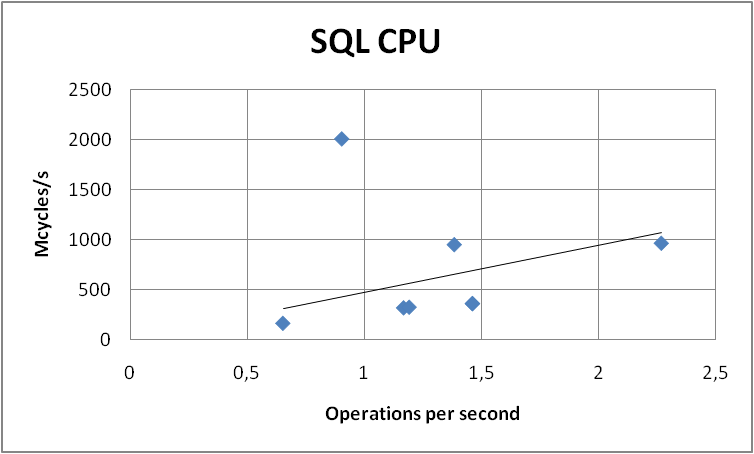
Read disk operations in the Web Front-End role are negligible, while write operations can be referred mainly to log writes.

##### Application Server



Read disk operations in the Application Server role are negligible, while write operations can be referred mainly to log writes.

##### SQL Server



### Approve Timesheets

This is a write operation that is managed through the Timesheet queue in Project Server. For this reason, we split the test in two parts: the first one is the user interface and en-queue operation, the second one is the de-queue operation. In order to separate the two parts, we stopped the queue service on the application server while approving timesheets, and for the second part we restarted the queue service without approving any other timesheet.

#### Input parameters

By using data profiles 1, 2, 5, and 9 (see "Data Profiles" above), the following parameters were varied across different test runs:

* Number of projects
* Number of visible projects per user
* Number of tasks per project
* Number of assignments per resource

None of the parameters above influenced measured data significantly. This was expected because the timesheet approval operations refer to the timesheet entity regardless of how many assignments are inside.

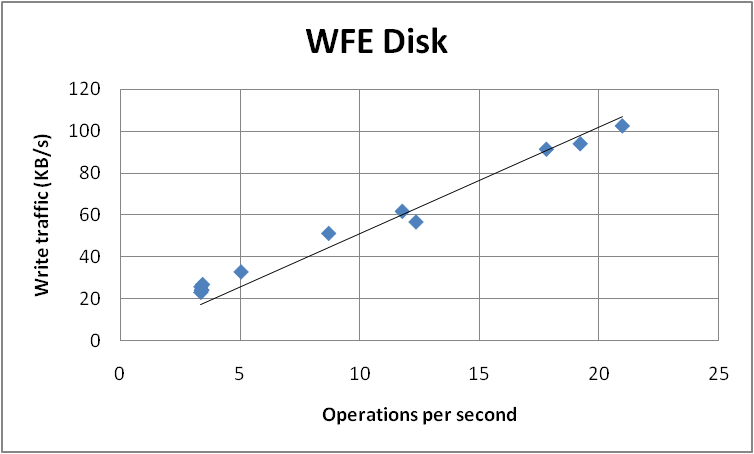
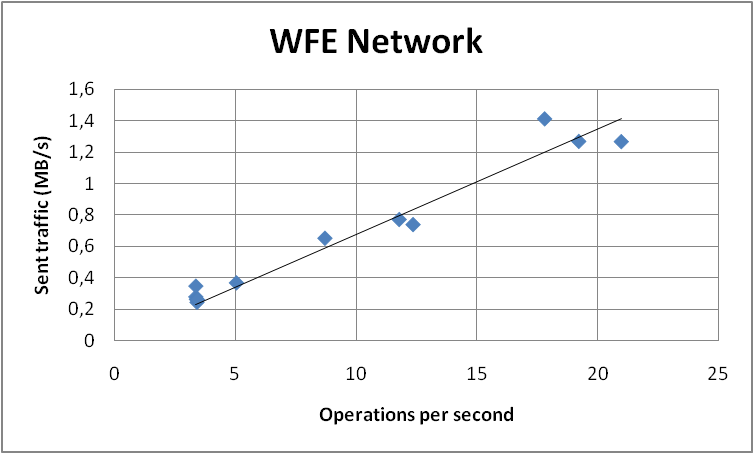
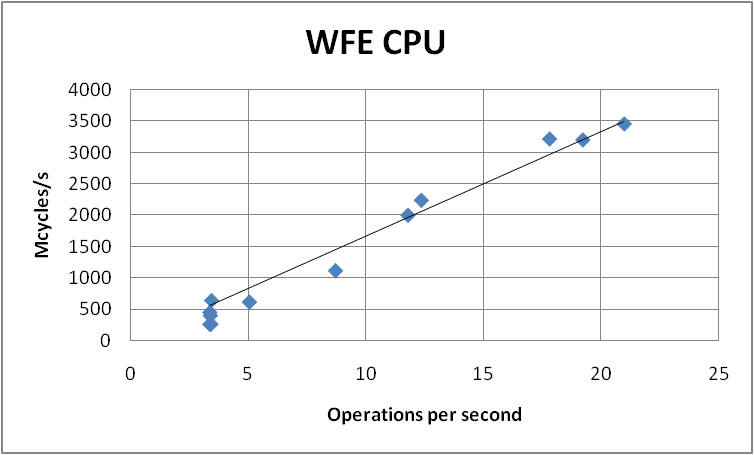
Moreover, the number of concurrent users was varied (20 to 50), leading to different throughputs obtained across the tests.

#### Results

In terms of throughput, we observed average results ranging from 3.4 to 21 operations per second for the user interface and en-queue operations. The de-queue operation throughputs ranged from 20.6 to 35.6 operations per second.

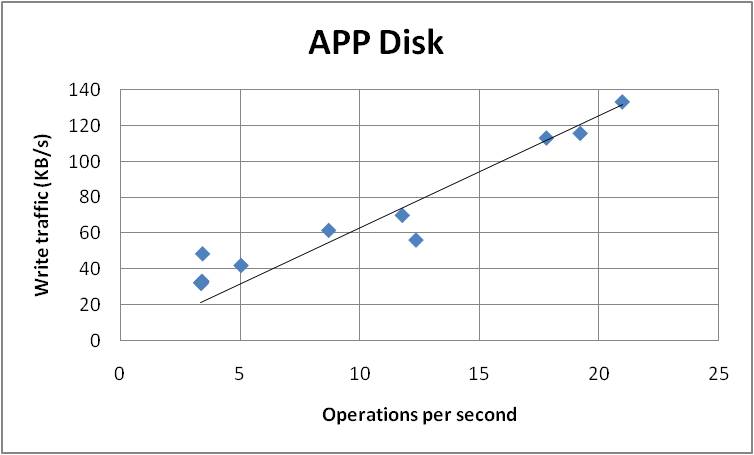
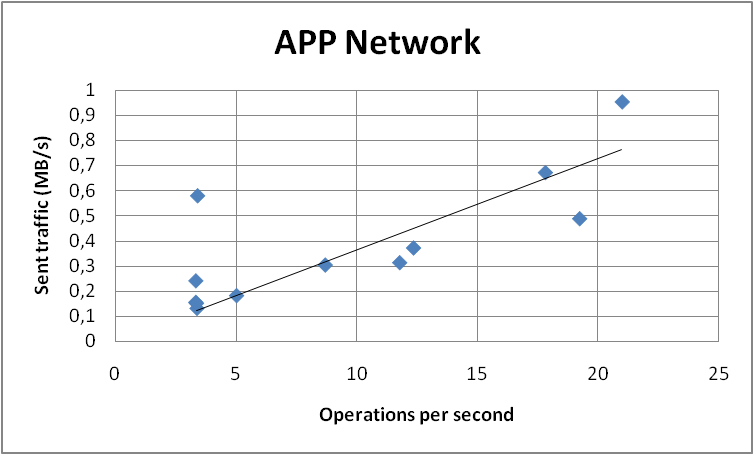
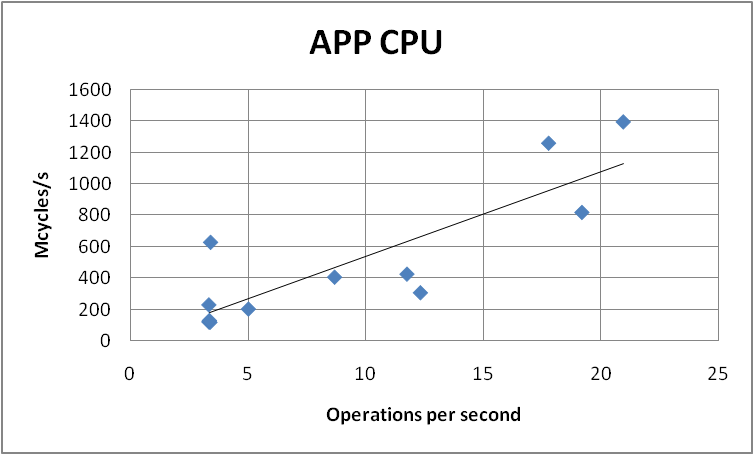
The following charts show the measured operation costs, and the estimated best-fit lines, as functions of throughput.

##### Web Front-End Server – User interface + en-queue operations

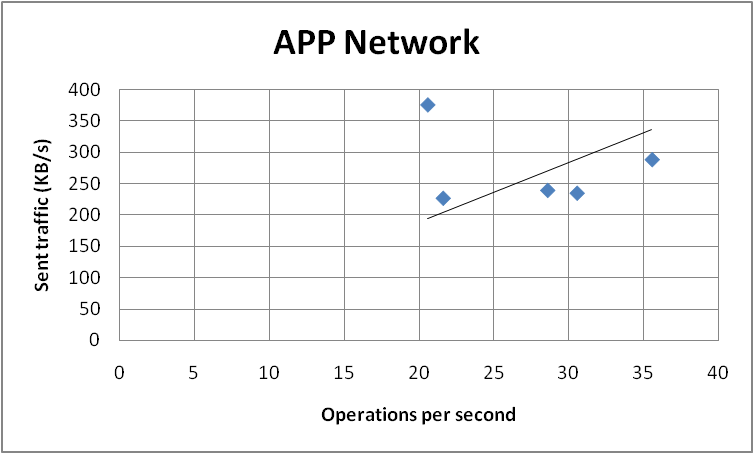
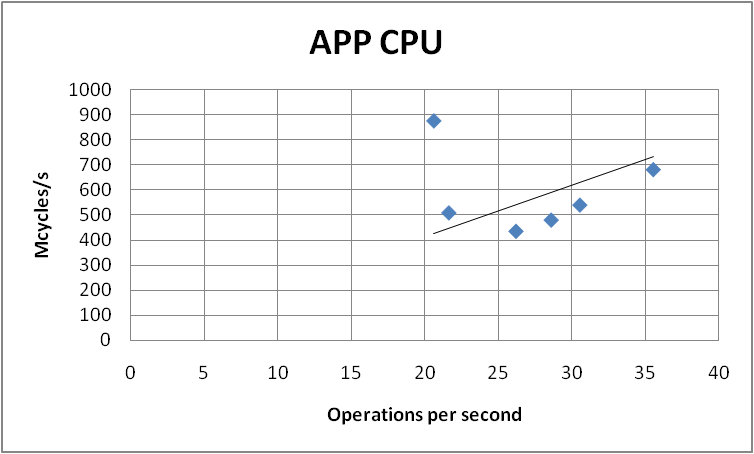


Read disk operations in the Web Front-End role are negligible, while write operations can be referred mainly to log writes.

##### Application Server – User interface + en-queue operations

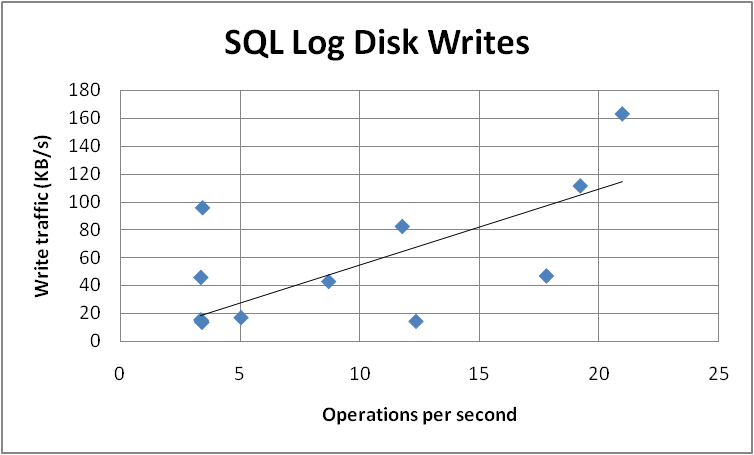
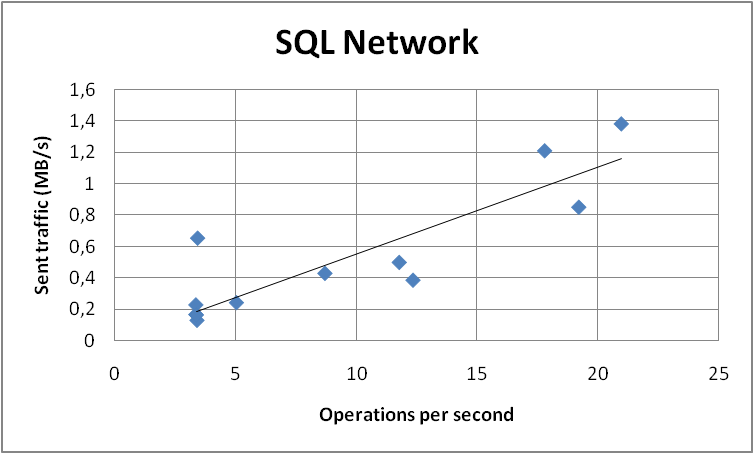
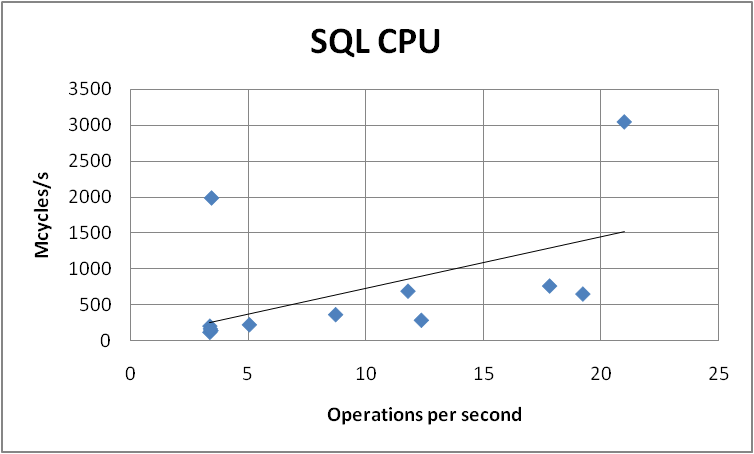


##### Application Server – De-queue operations

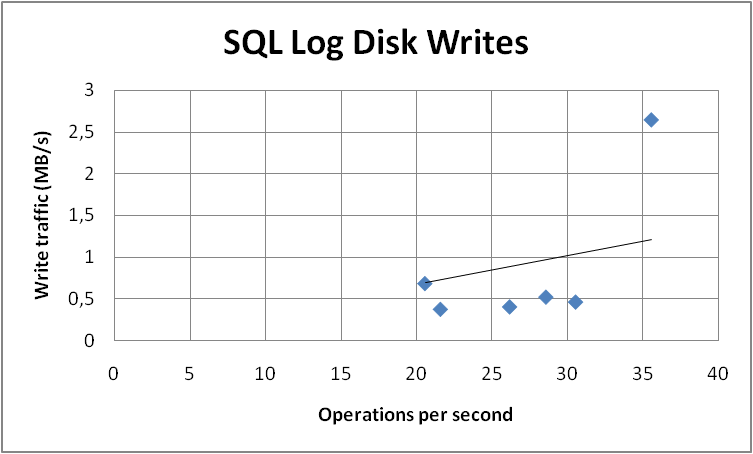
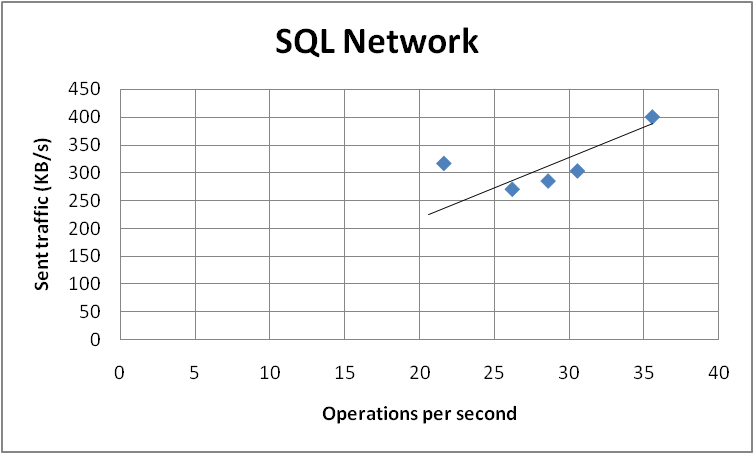
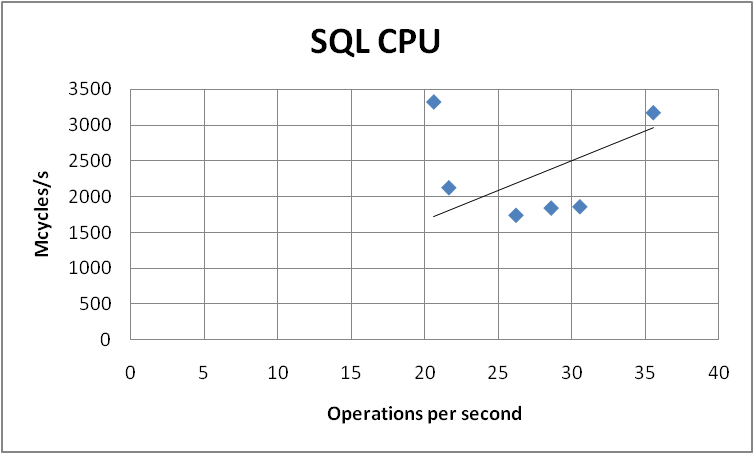


In both cases, read disk operations in the Application Server role are negligible. Write operations can be referred mainly to log writes for user interface and en-queue operations, and are negligible for the de-queue operations.

##### SQL Server – User interface + en-queue operations



##### SQL Server – De-queue operations



### Open Projects

#### Input parameters

By using data profiles 2, 3, 5, and 9 (see "Data Profiles" above), the following parameters were varied across different test runs:

* Number of projects
* Number of visible projects per user
* Number of tasks per project
* Number of assignments per resource

None of the parameters above influenced measured data significantly in our tests.

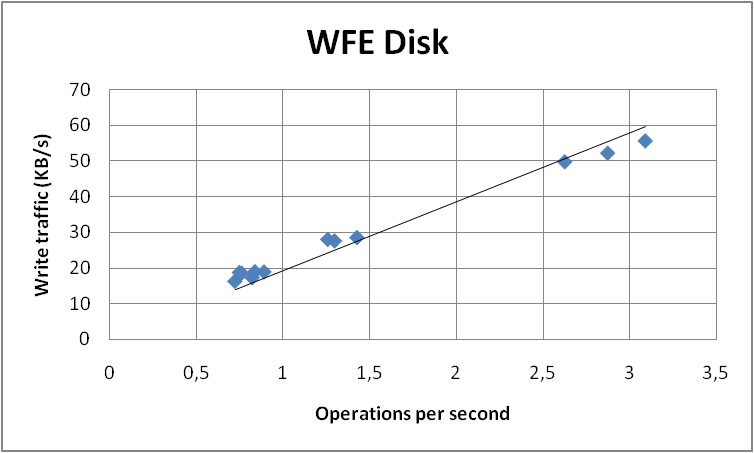
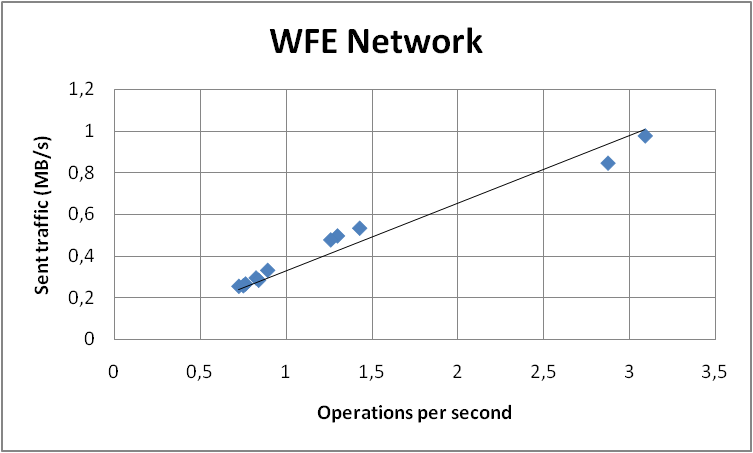
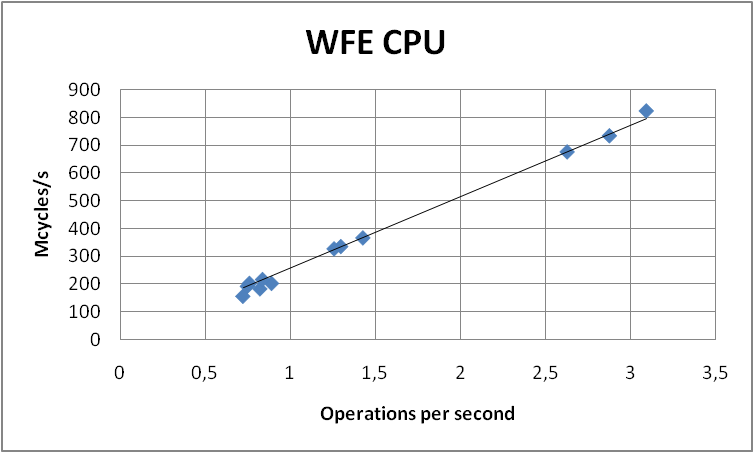
Moreover, the number of concurrent user sessions was varied (6 to 24), leading to different throughputs obtained across the tests.

#### Results

In terms of throughput, we observed average results ranging from 0.7 to 3.1 operations per second.

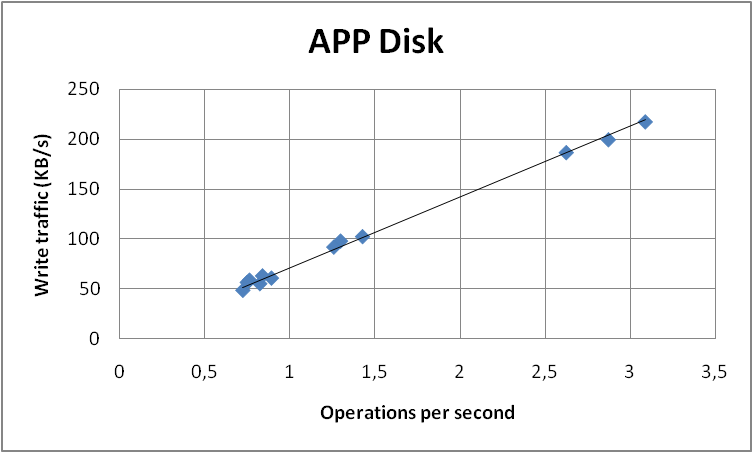
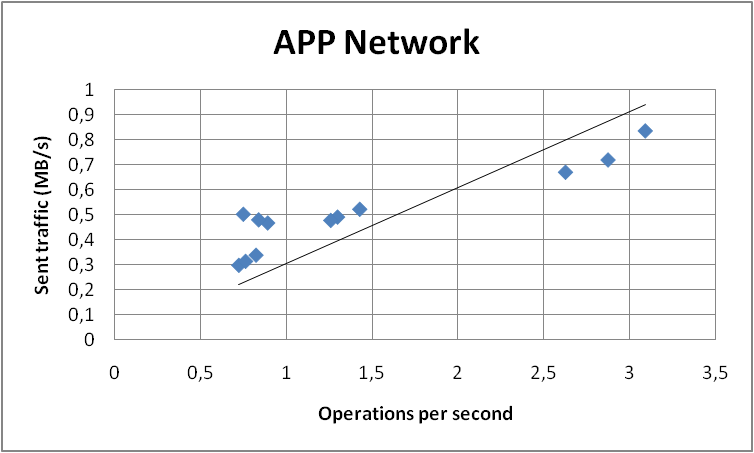
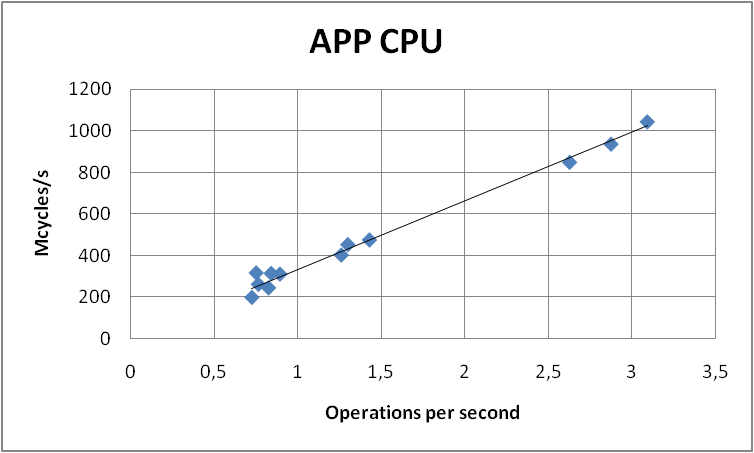
The following charts show the measured operation costs, and the estimated best-fit lines, as functions of throughput.

##### Web Front-End Server



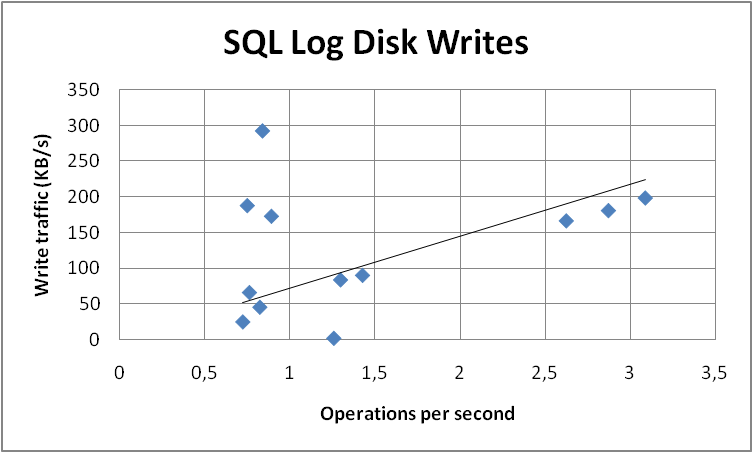
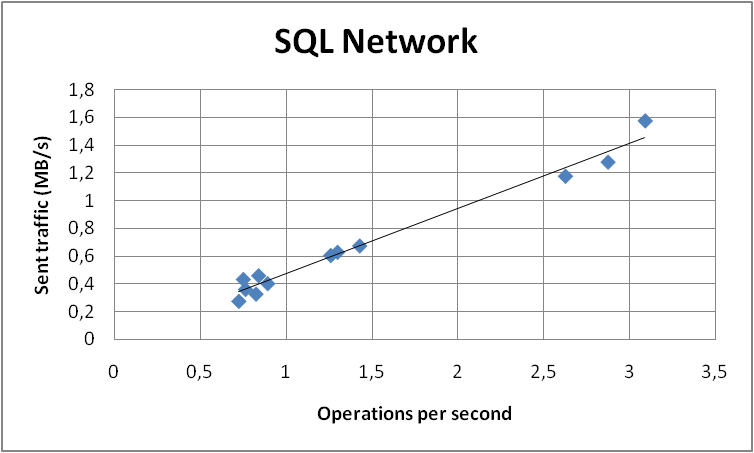
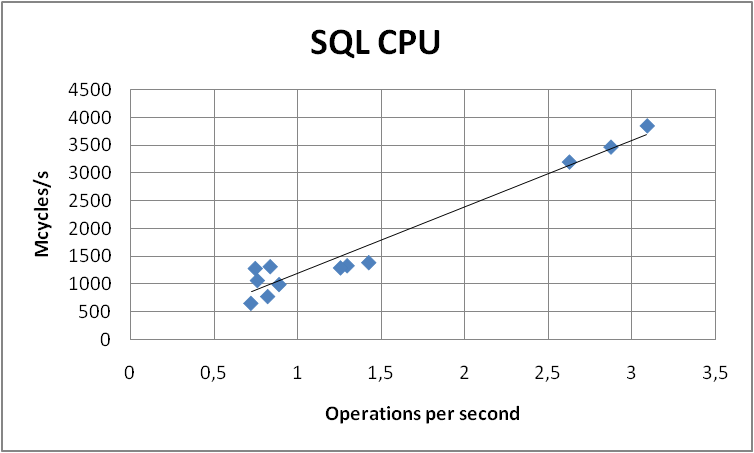
Read disk operations in the Web Front-End role are negligible, while write operations can be referred mainly to log writes.

##### Application Server



Read disk operations in the Application Server role are negligible, while write operations can be referred mainly to log writes.

##### SQL Server



### Save Projects

This is a write operation that is managed through the Project queue in Project Server. For this reason, we split the test in two parts: the first one is the user interface and en-queue operation, the second one is the de-queue operation. In order to separate the two parts, we stopped the queue service on the application server while saving projects, and for the second part we restarted the queue service without saving any other project.

#### Input parameters

By using data profiles 1, 2, 5, and 9 (see "Data Profiles" above), the following parameters were varied across different test runs:

* Number of projects
* Number of visible projects per user
* Number of tasks per project
* Number of assignments per resource

None of the parameters above influenced measured data significantly in our tests.

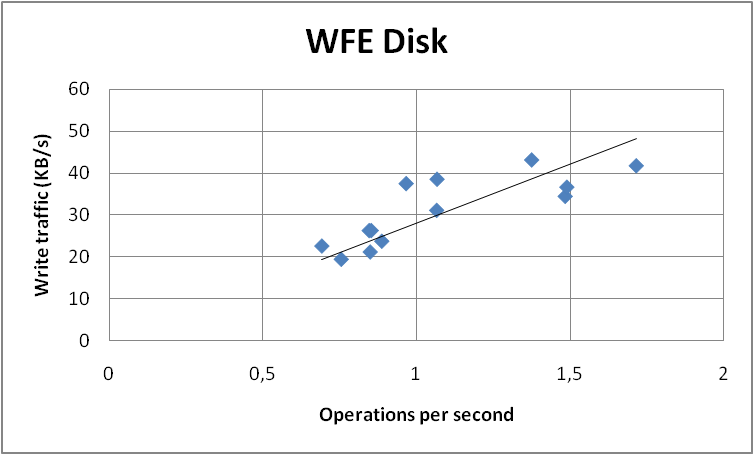
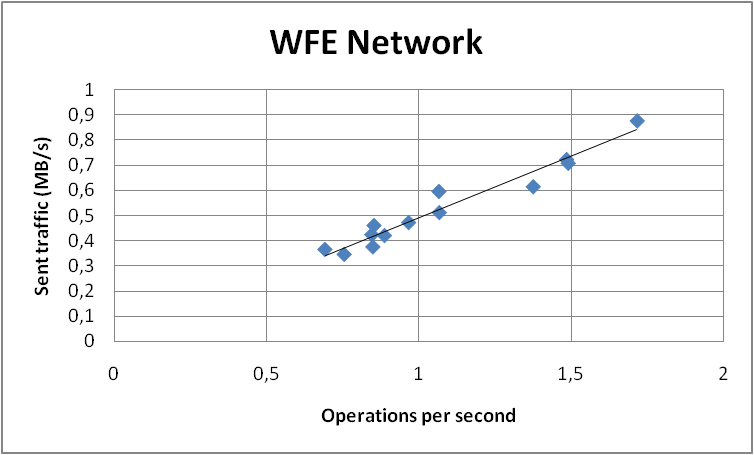
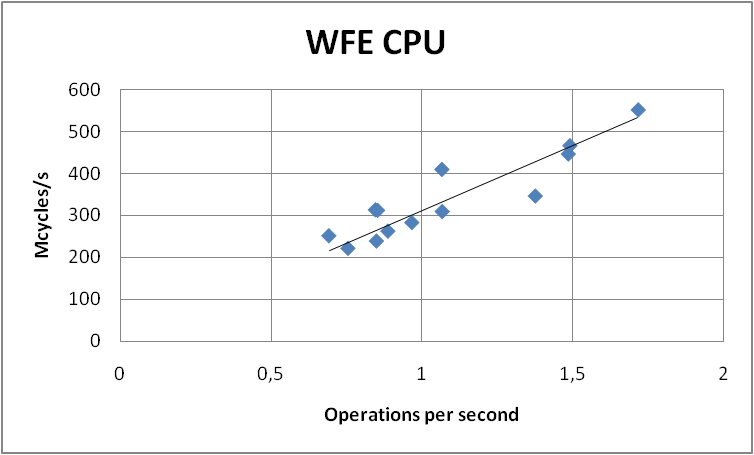
Moreover, the number of concurrent user sessions was varied (10 to 24), leading to different throughputs obtained across the tests.

#### Results

In terms of throughput, we observed average results ranging from 0.7 to 1.7 operations per second for the user interface and en-queue operations. The de-queue operation throughputs ranged from 0.8 to 3.3 operations per second.

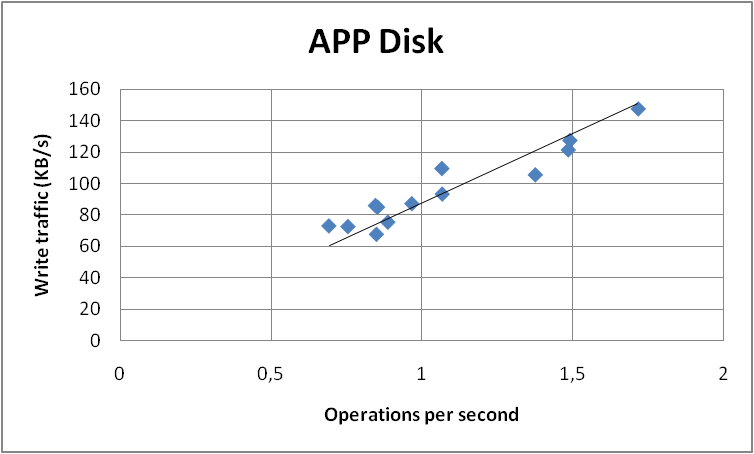
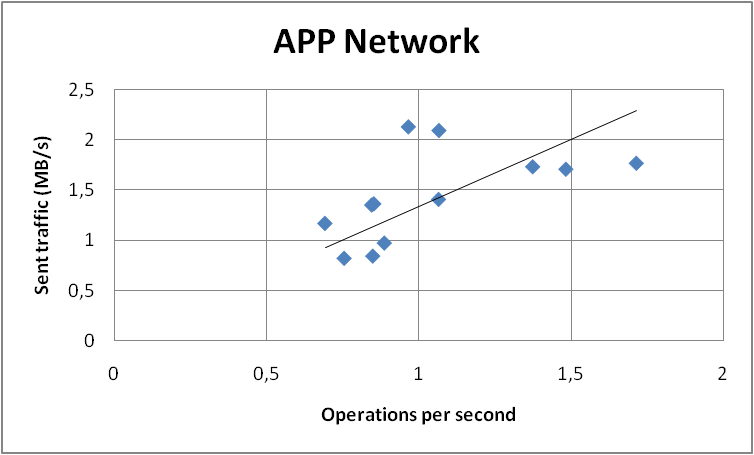
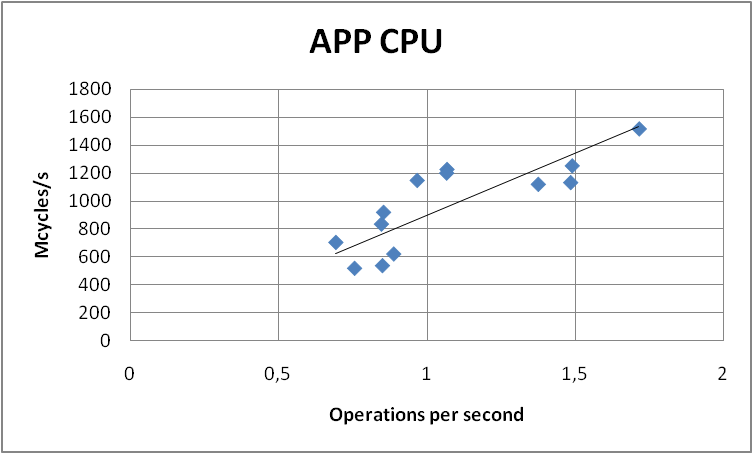
The following charts show the measured operation costs, and the estimated best-fit lines, as functions of throughput.

##### Web Front-End Server – User interface + en-queue operations

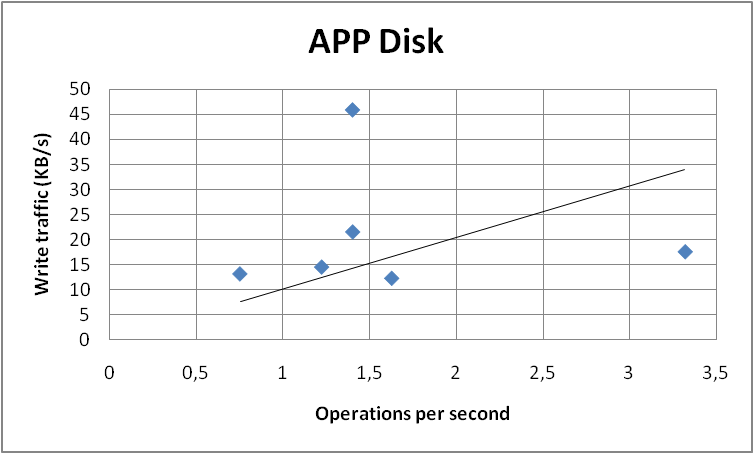
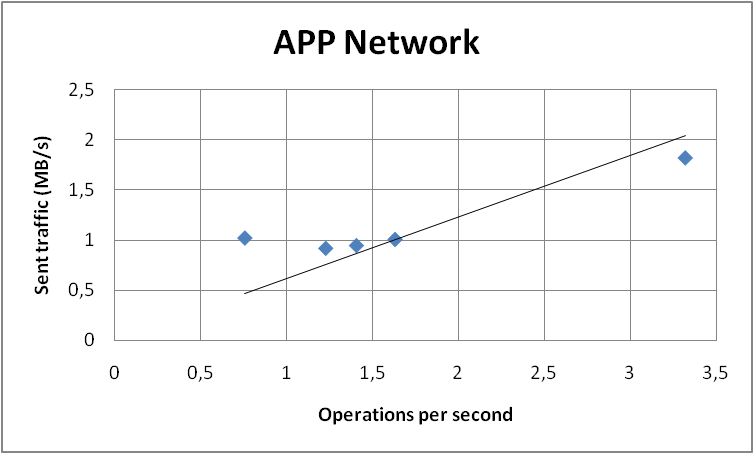
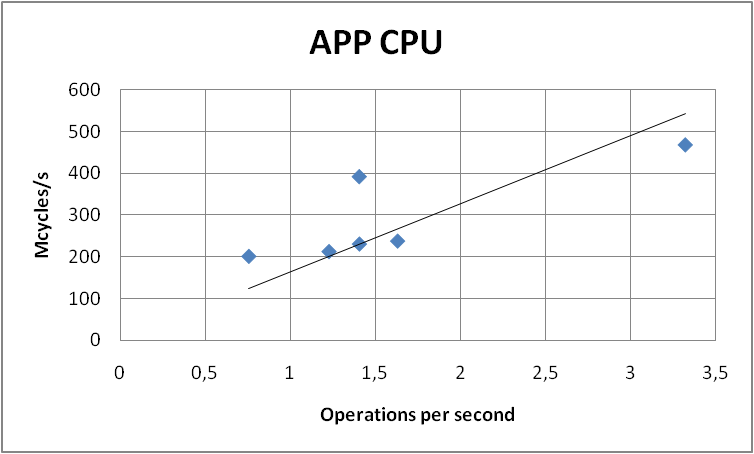


Read disk operations in the Web Front-End role are negligible, while write operations can be referred mainly to log writes.

##### Application Server – User interface + en-queue operations

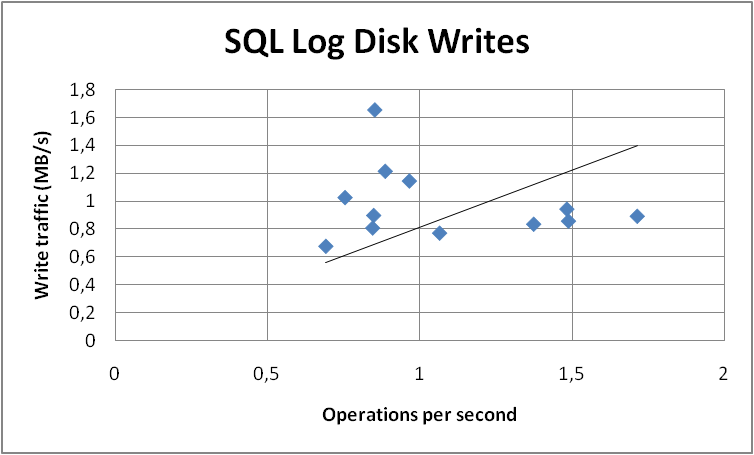
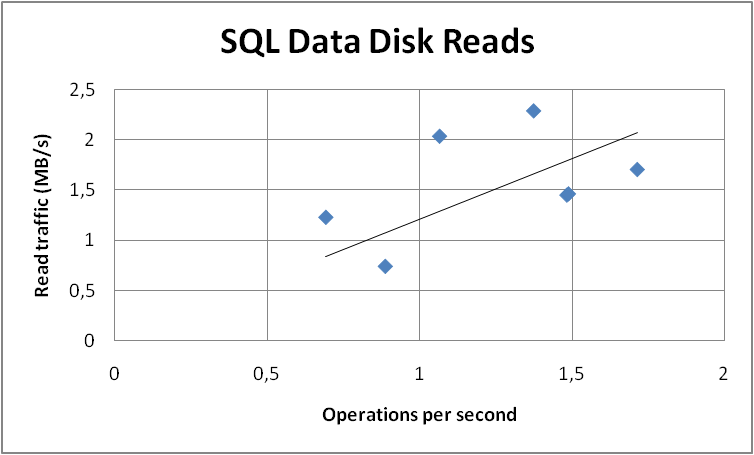
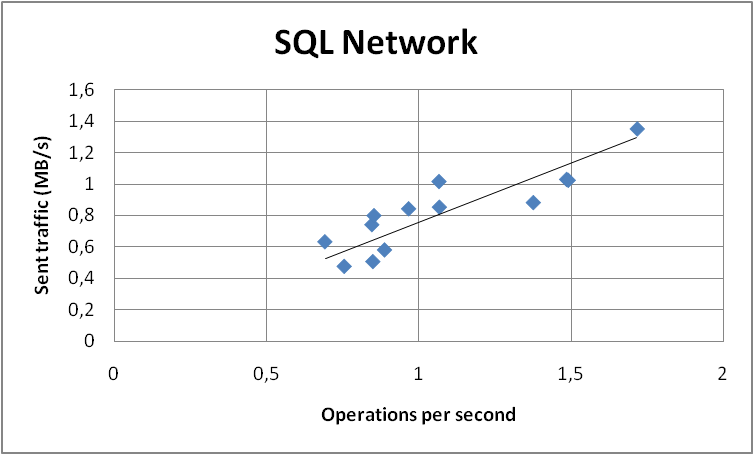
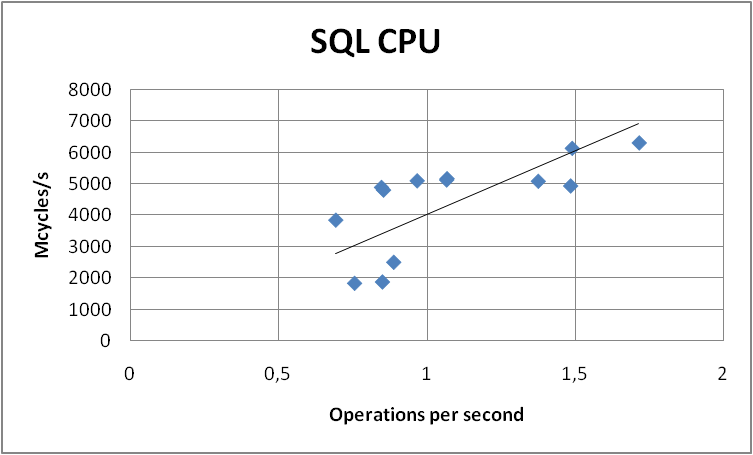


##### Application Server – De-queue operations

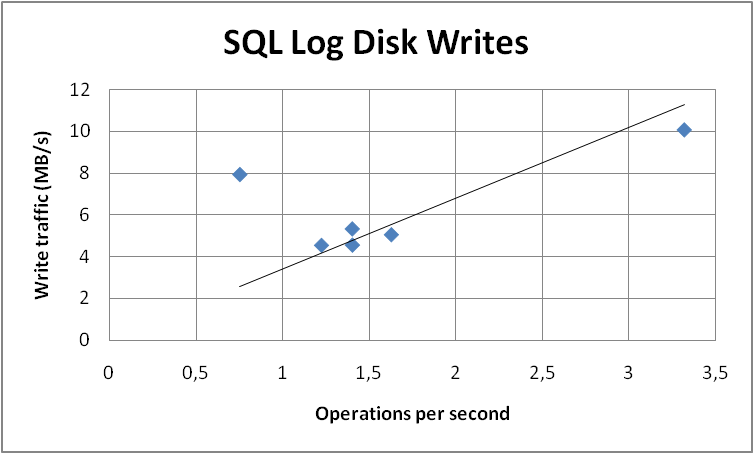
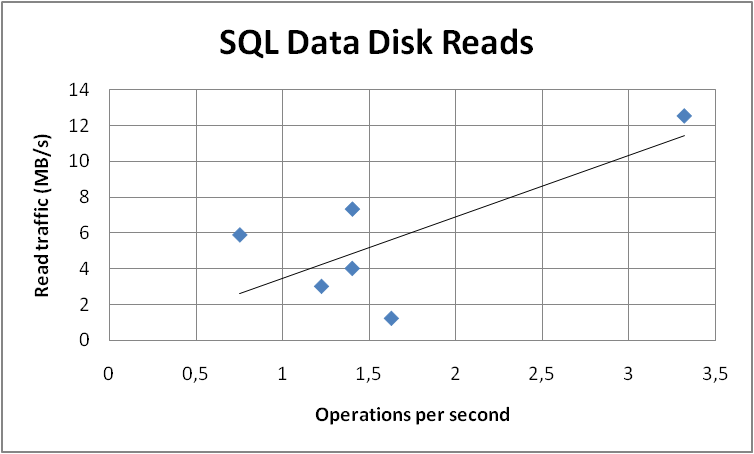
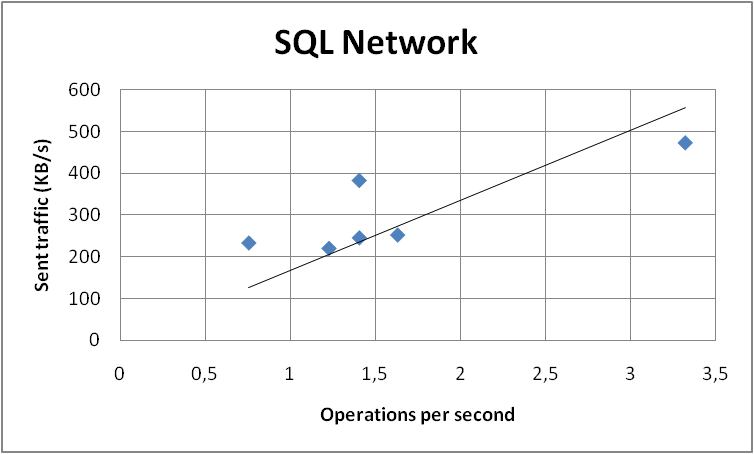
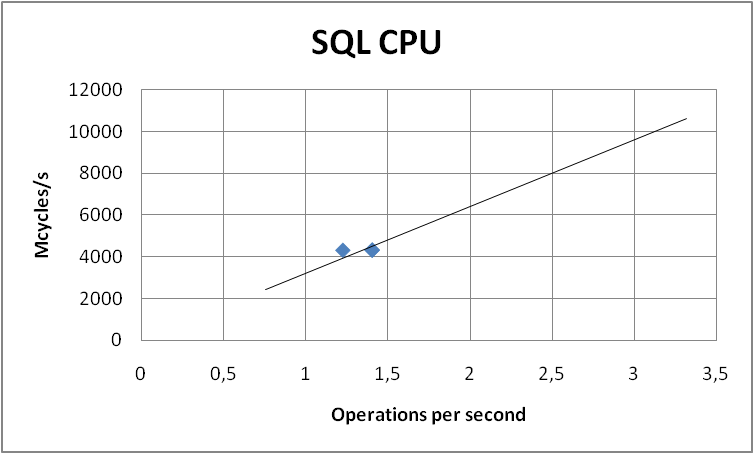


Read disk operations in the Application Server role are negligible, while write operations can be referred mainly to log writes.

##### SQL Server – User interface + en-queue operations



##### SQL Server – De-queue operations



### Publish Projects

This is a write operation that is managed through the Project queue in Project Server. For this reason, we split the test in two parts: the first one is the user interface and en-queue operation, the second one is the de-queue operation. In order to separate the two parts, we stopped the queue service on the application server while publishing projects, and for the second part we restarted the queue service without publishing any other project.

#### Input parameters

By using data profiles 1, 2, 5, and 9 (see "Data Profiles" above), the following parameters were varied across different test runs:

* Number of projects
* Number of visible projects per user
* Number of tasks per project
* Number of assignments per resource

None of the parameters above influenced measured data significantly. This was expected because the timesheet approval operations refer to the timesheet entity regardless of how many assignments are inside.

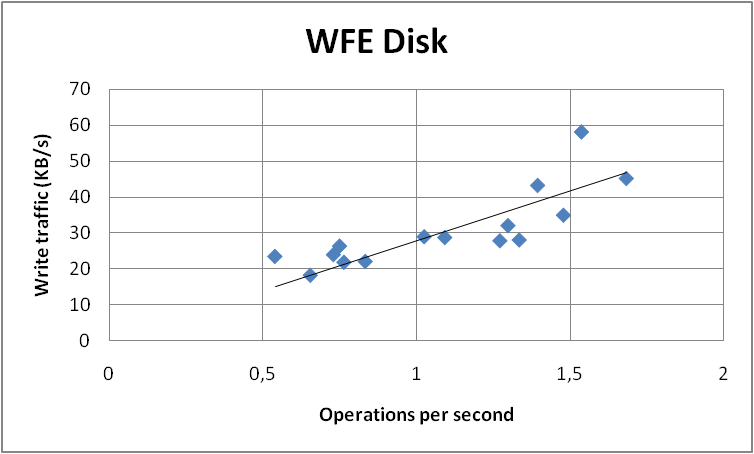
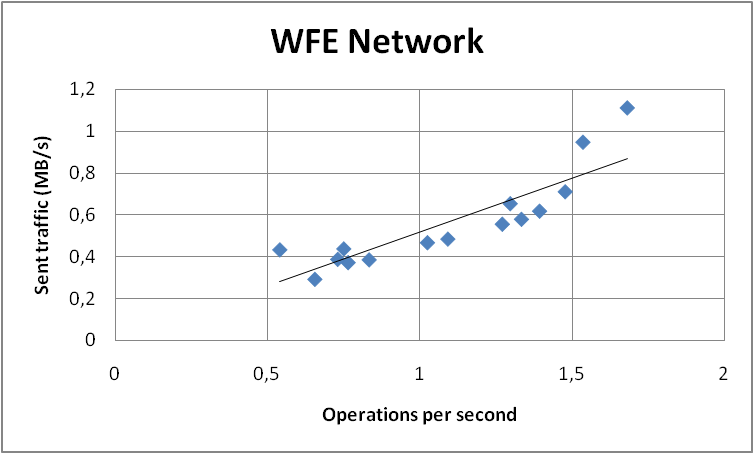
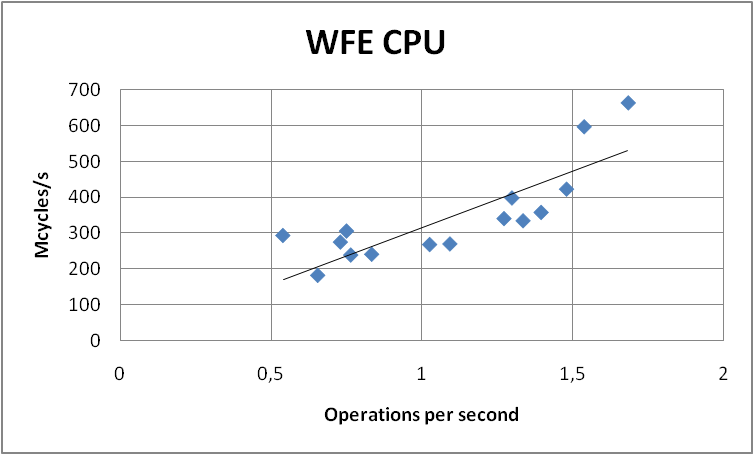
Moreover, the number of concurrent users was varied (6 to 24), leading to different throughputs obtained across the tests.

#### Results

In terms of throughput, we observed average results ranging from 0.5 to 1.7 operations per second for the user interface and en-queue operations. The de-queue operation throughputs ranged from 0.4 to 1.2 operations per second.

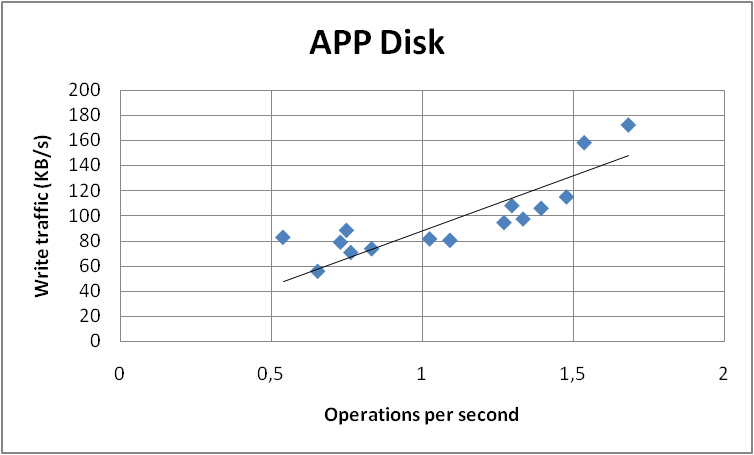
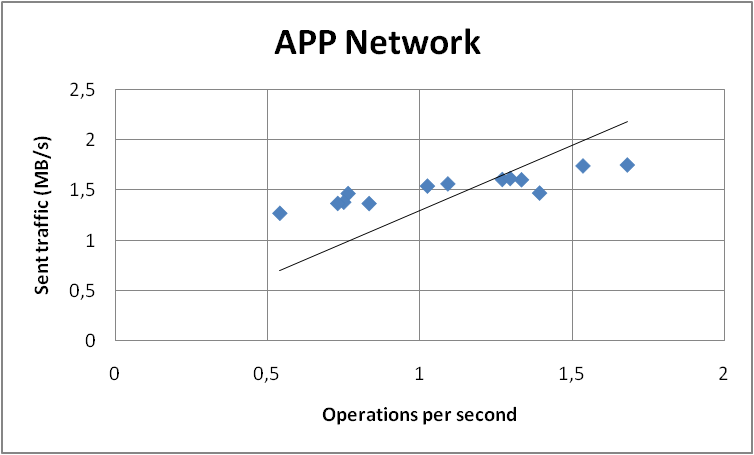
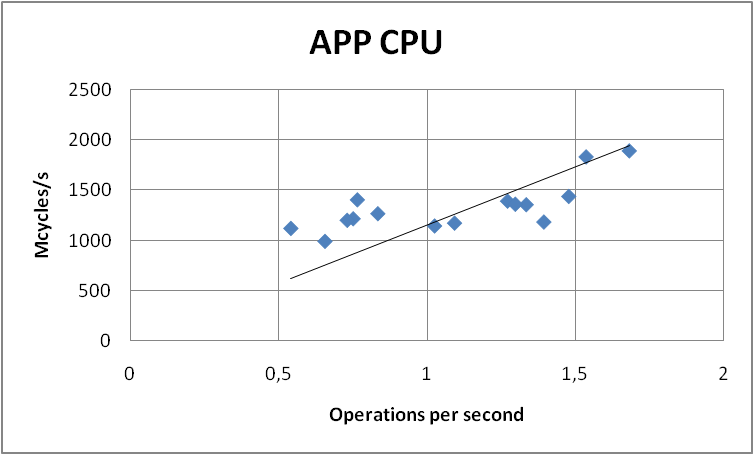
The following charts show the measured operation costs, and the estimated best-fit lines, as functions of throughput.

##### Web Front-End Server – User interface + en-queue operations

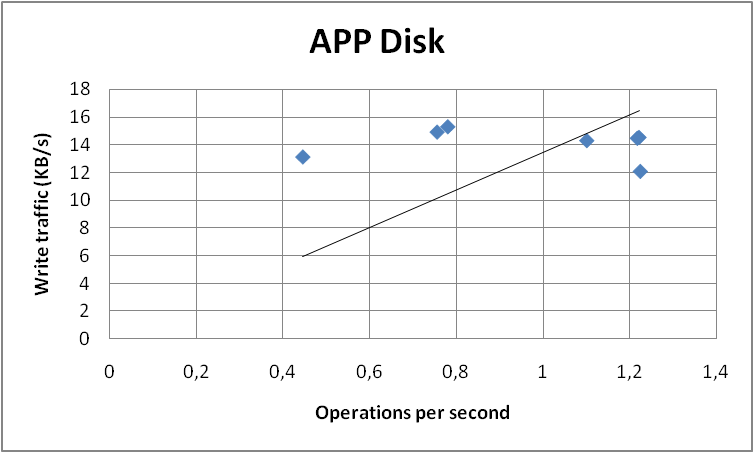
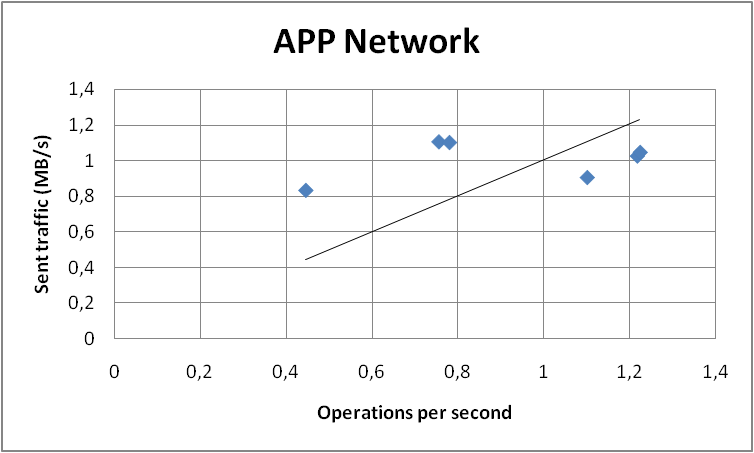
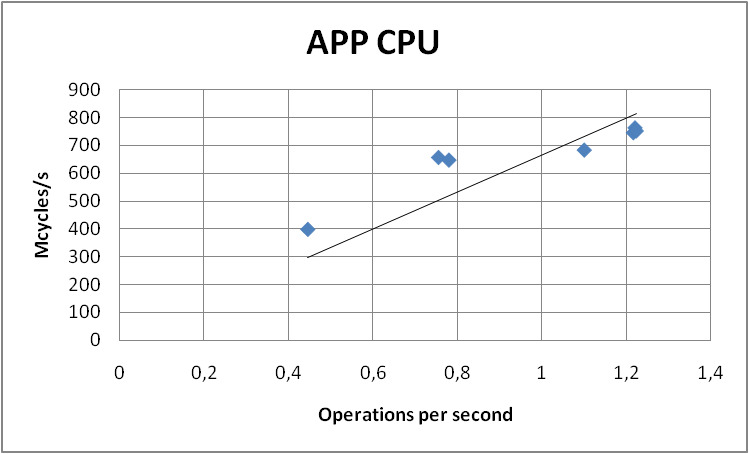


Read disk operations in the Web Front-End role are negligible, while write operations can be referred mainly to log writes.

##### Application Server – User interface + en-queue operations

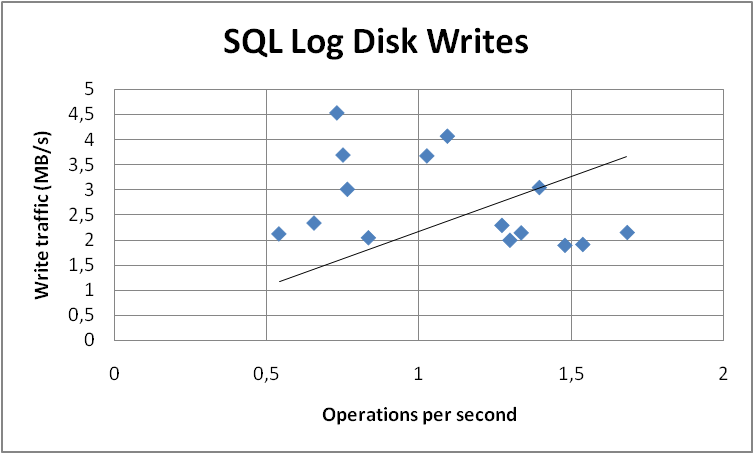
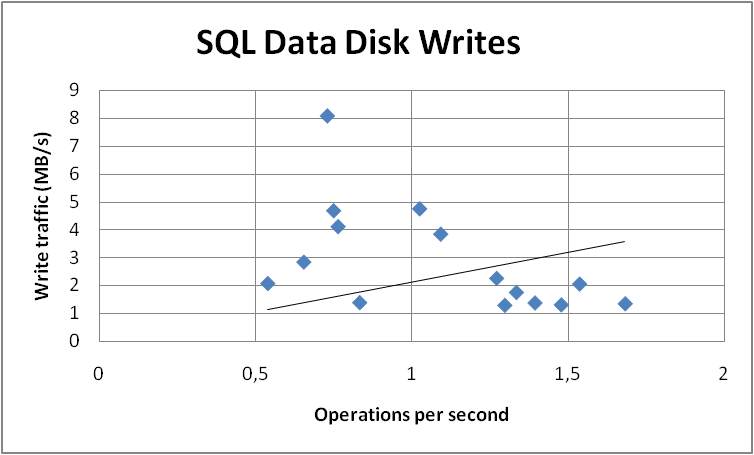
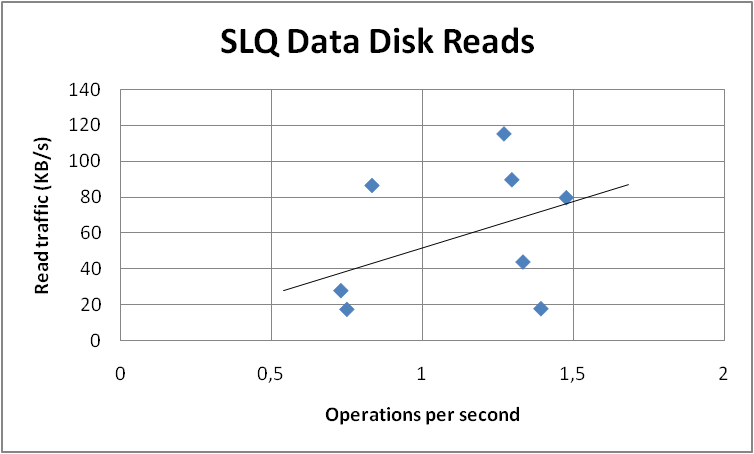
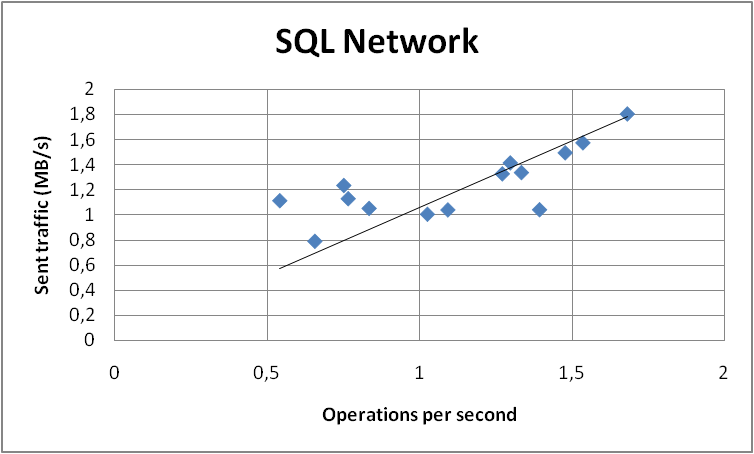
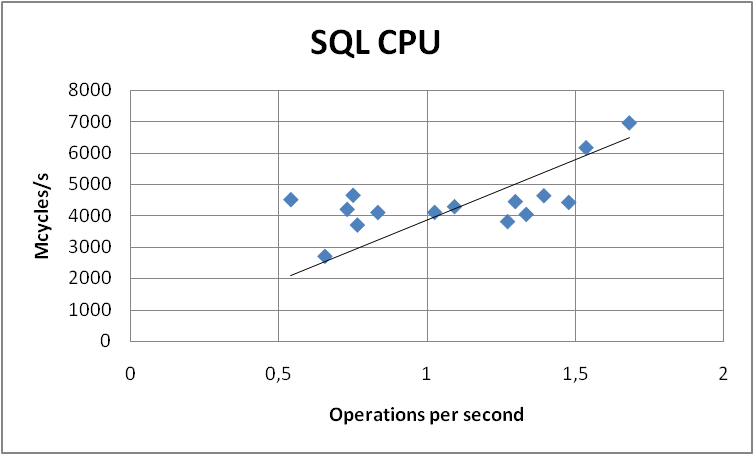


##### Application Server – De-queue operations

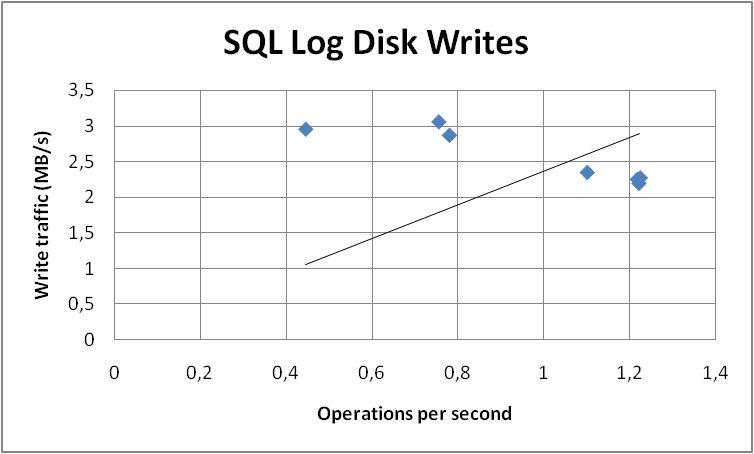
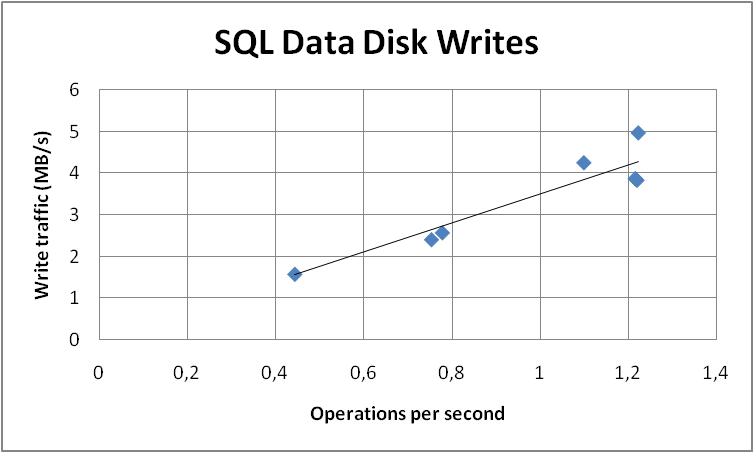
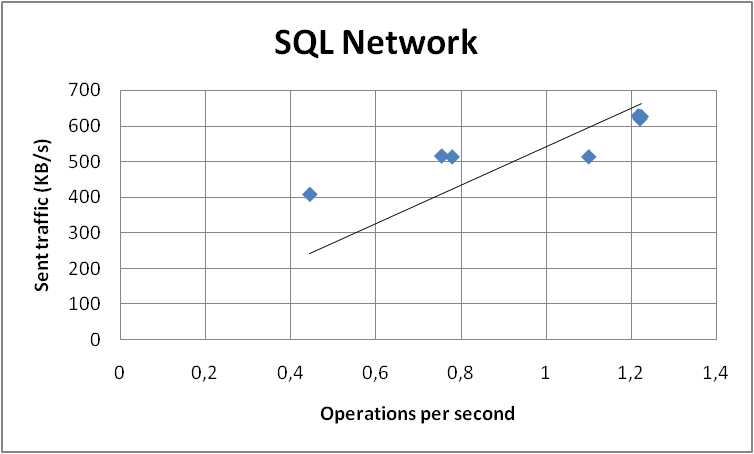
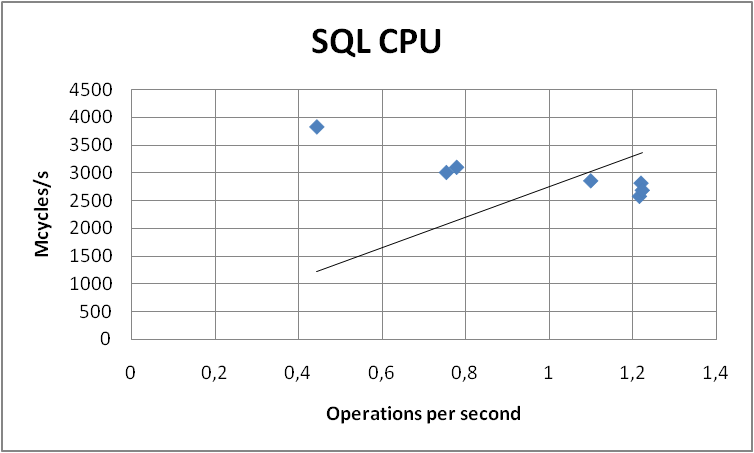


Read disk operations in the Application Server role are negligible, while write operations can be referred mainly to log writes.

##### SQL Server – User interface + en-queue operations



##### SQL Server – De-queue operations



Read operations on the data disk for de-queue operations were negligible.

## Mixed Load Tests

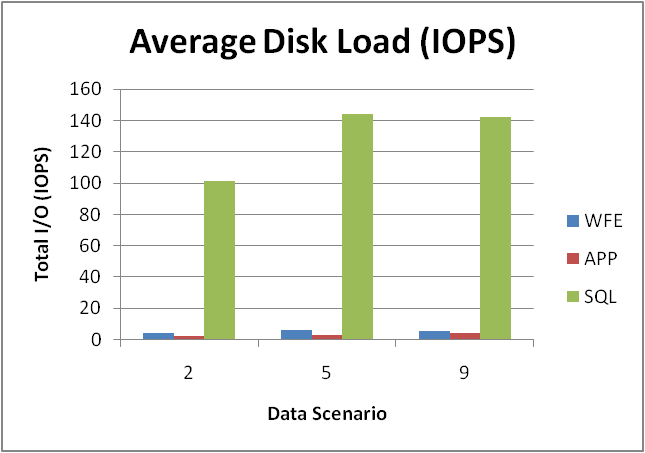
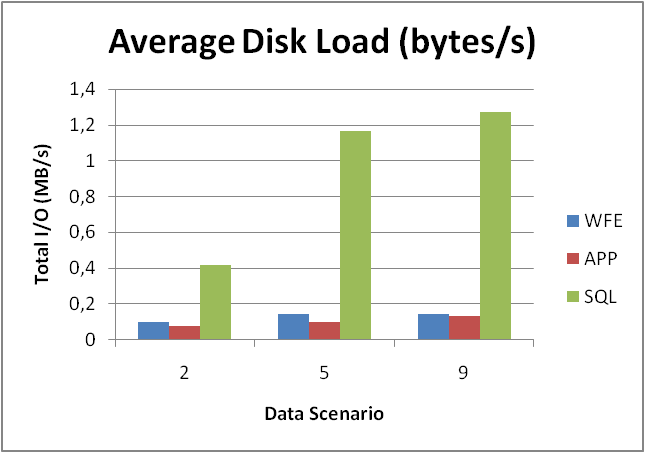
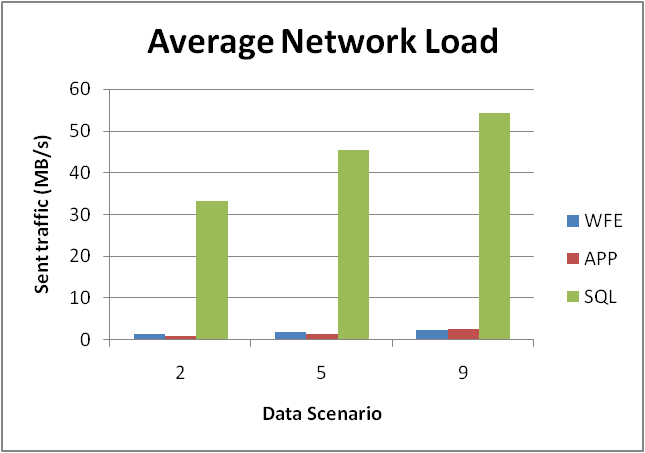
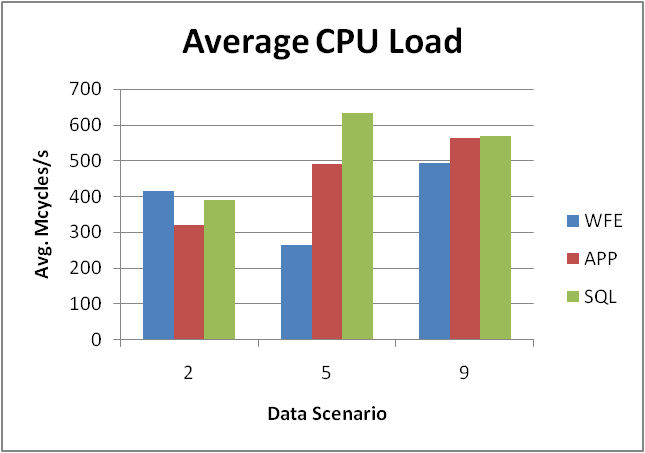
These tests were designed to mix all the operations that were previously tested independently in Visual Studio Team System 2008 Test Edition, in a realistic usage profile mix. The mix of operations that has been considered is the following:

|  |  |  |
| --- | --- | --- |
| Operation | Test | Percentage |
| Access PWA home page | PWAHomePage web test | 20 |
| Access Project Center views | ProjectCenterPage web test | 15 |
| Access Resource Center views | ResourceCenterPage web test | 8 |
| Submit Task Updates | MyTasksPage web test | 8 |
| SubmitTaskUpdates unit test | 4 |
| Submit Timesheets | MyTimesheetPage web test | 14 |
| TimesheetCreateFillSubmit unit test | 7 |
| Approve Task Updates | TaskUpdateApprovalPage web test | 8 |
| ApproveTaskUpdate unit test | 4 |
| Approve Timesheets | TimesheetApprovalPage web test | 8 |
| ApproveTimesheets unit test | 4 |

Several test runs have been performed, changing the underlying data profile and the server topology in the farm, in order to understand how the load measured on the systems changes in a realistic mixed usage profile.

### Testing Against Multiple Data Profiles

The following graphs show the average costs calculated from the measures taken on several mixed tests runs. The server topology was kept constant across these tests (1 Web Front-End, 1 Application Server).



### Testing Against Multiple Server Topologies

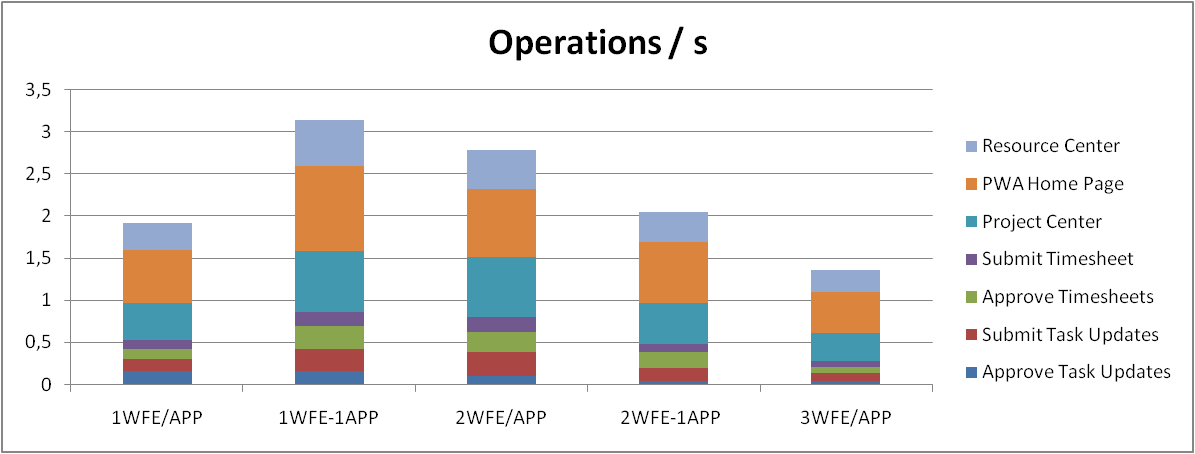
The server farm topologies that had been tested in the lab were the following:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Server 1 | Server 2 | Server 3 |
| 1WFE-1APP | Web Front-End | Application Server | - |
| 2WFE-1APP | Web Front-End (NLB) | Web Front-End (NLB) | Application Server |
| 1WFE/APP | Web Front-End Application Server | - | - |
| 2WFE/APP | Web Front-End (NLB) Application Server | Web Front-End (NLB) Application Server | - |
| 3WFE/APP | Web Front-End (NLB) Application Server | Web Front-End (NLB) Application Server | Web Front-End (NLB) Application Server |

Several mixed tests were run with the different topologies described above, while the data profile was kept constant (data profile 9, see "Data Profiles" above).

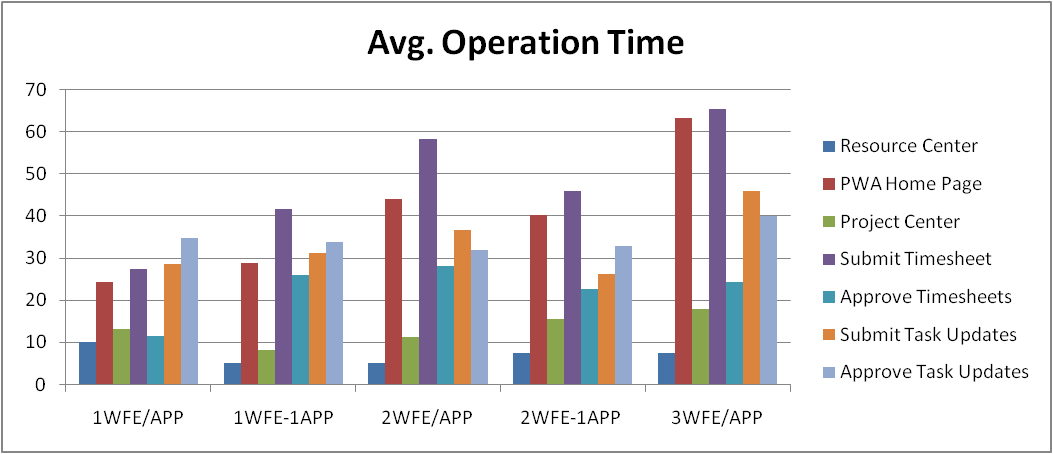
#### Throughput

The following graph shows the average results in terms of global throughput (average number of operations completed successfully per second).



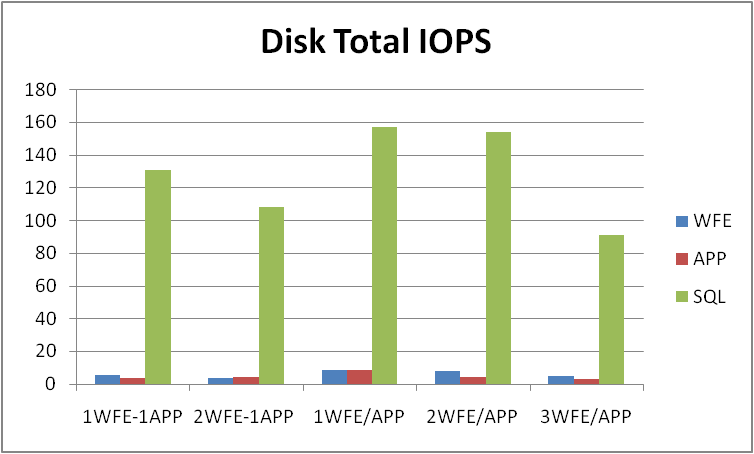
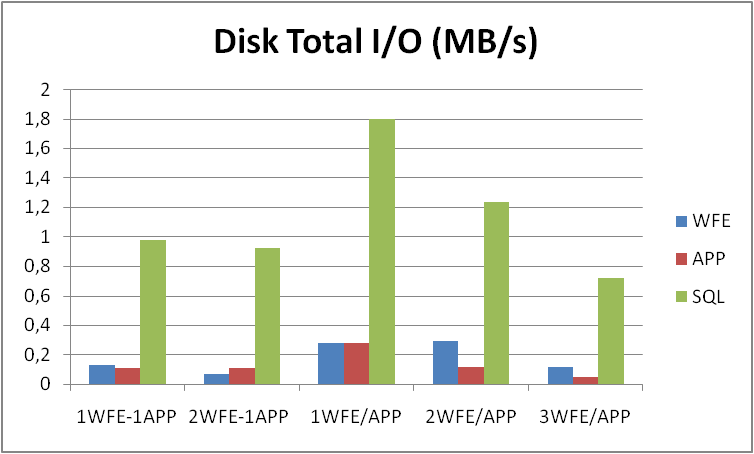
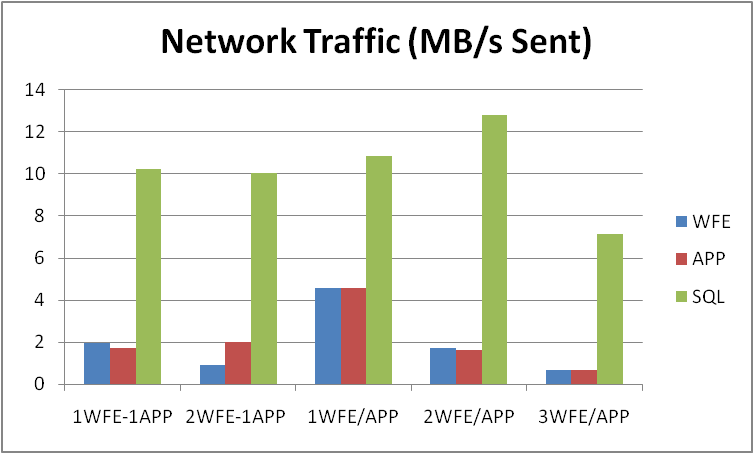
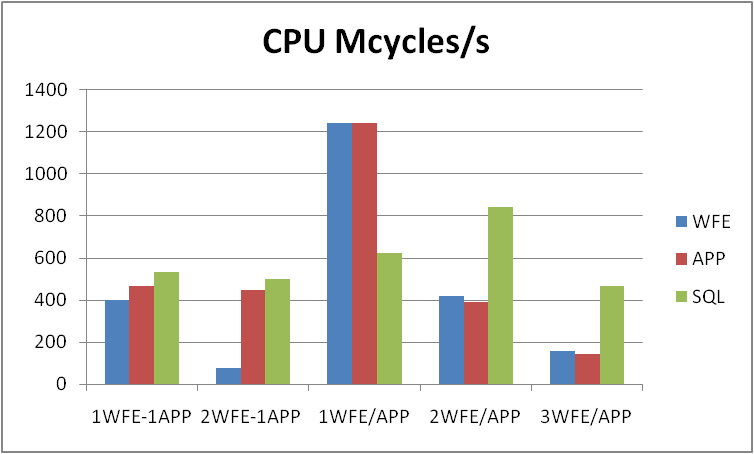
#### Operation Time

In terms of total time to complete the operations, the following graph shows overall average results across all the mixed tests under load.



#### Server Load

The following graphs show the load generated on the individual servers when running the mixed tests across different topologies.



1. Once the data population process has finished saving/publishing projects, you should allow enough time before performing the backup to ensure the project queue has been drained completely. [↑](#footnote-ref-2)