Project #4 - Remote Key/Value Database

Version 1.4, Revised: 10/16/2015 06:12:17
Due Date: Wednesday November 18th

Purpose:

In this project we will explore how to use the noSQL database we built in Project #2 from remote clients.

- One or more client(s) will concurrently supply the database with a stream of inputs through an input queue the database server provides.
- One or more client(s) will concurrently extract data from the database by enqueuing requests and waiting for replies.

An important part of this project is to assess the performance of the database and communication as the number of concurrent reader and writer clients increases.

Requirements:

Your Remote Key/Value Data Storage project:

1. **Shall** be implemented in C# using the facilities of the .Net Framework Class Library and Visual Studio 2015, as provided in the ECS clusters. All communication between processes and machines **shall** be implemented using Windows Communication Foundation (WCF) and all client display **shall**be implemented with Windows Presentation Foundation (WPF).
2. (1) **Shall** use the noSQL database you implemented in Project #2[1].
3. (3) **Shall** use WCF to communicate between clients and a server that exposes the noSQL database through messages that are sent by clients and enqueued by the server. Each message **shall** be processed by the server to interact with the database and results are sent, as messages, back to queues provided by each client requestor.
4. (3) **Shall** demonstrate that the required operations implemented in Project #2 can be executed remotely, e.g.:
    - add, delete, and edit key/value pairs
    - persist and restore the database from an XML file
    - support the same queries as required in Project #2

    These operation requests **shall** be sent to the remote database in the form of messages, described by a WCF Data Contract[2]. Replies **shall** be returned to the requestor in the form of WCF messages, using a suitable Data Contract[2].
5. (3) Write clients - those that send data to the remote database - **shall** use may use either a console interface or a WPF graphical user interface, and**shall** provide a stream of messages as described below. The content of several different messages will be defined in an XML file, read at client startup. The number of messages sent of each type is defined by one or more elements of the XML file[3]. After reading the XML file the write client starts a high-resolution timer, sends the messages, stops the timer and writes the elapsed time to the console or GUI.
6. (1) Each write client **shall** accept an option switch on the command line that determines whether messages are logged to the console or GUI as they are sent[4].
7. (3) Read clients - those that send query requests to the remote database - **shall** provide a response queue for incoming response messages and display

results in a WPF interface. The content of several different request messages will be defined in an XML file, read at client startup[3].

8. (3) Read clients **shall** provide options to:
    - display some part of each response as it is processed (what to display is left up to you)
    - set a specified number of requests to send, start a high-resolution timer when the first request is sent and stop the timer when the last request is received. Think of this as a "self-test" option.
9. (2) **Shall** be accompanied by a test executive[5] that clearly demonstrates you've met all the functional requirements #2-#7, above. If you do not demonstrate a requirement you will not get credit for it even if you have it correctly implemented.
10. (1) The test executive **shall** accept from the command line the number of writers and readers to start[6].

---

1. You may use the instructor's solution if your Project #2 results are not fully functional. However, you will only get credit for your contributed value. While it is certainly possible to get an excellent grade using the instructor's solution, it is easier to do so if you can demonstrate that your Project #2 results are good enough too use here. Note that you may make any changes to your database implementation, or the instructor's, you need to in order to implement this project.
2. You may use more than one Data Contract for request messages and for reply messages if you think that you need to do that.
3. The intent here is that the XML file defines a sequence of elements, one for each message. These are read and a collection of prototype messages are created. The client then loops over this set of messages sending each one until the required number of messages has been sent.
4. I/O is slow, so we provide this option to disable for more accurate measurements of performance.
5. You may wish to provide three kinds of clients:
    - Console or GUI client that demonstrates requirments are met.
    - Write client, used for performance testing.
    - GUI read client, also used for performance testing.
6. This is fairly simple to accomplish using the System.Diagnostics.Process class from the .Net framework.

What you need to know:

In order to successfully meet these requirements you will need to:

1. Write C# code and use advanced facilities of the .Net Framework.
2. Understand .Net threading docs and Threading Examples.
3. Use WCF example code and the WCF Docs.
4. Use WPF example code and the WPF Docs.