# Principles of Computer System Design (PCSD), Block 2, 2012 / 2013

# Assignment 1

This assignment is due via Absalon on December 4, 23:55pm. This assignment is going to be evaluated pass-fail, as announced in the course description. While individual hand-ins will be accepted, we strongly recommend that this assignment be solved in groups of two students. Groups may at a maximum include three students.

A well-formed solution to this assignment should include a PDF file with answers to all exercises as well as questions posed in the programming part of the assignment. In addition, you must submit your code along with your written solution. Evaluation of the assignment will take both into consideration.

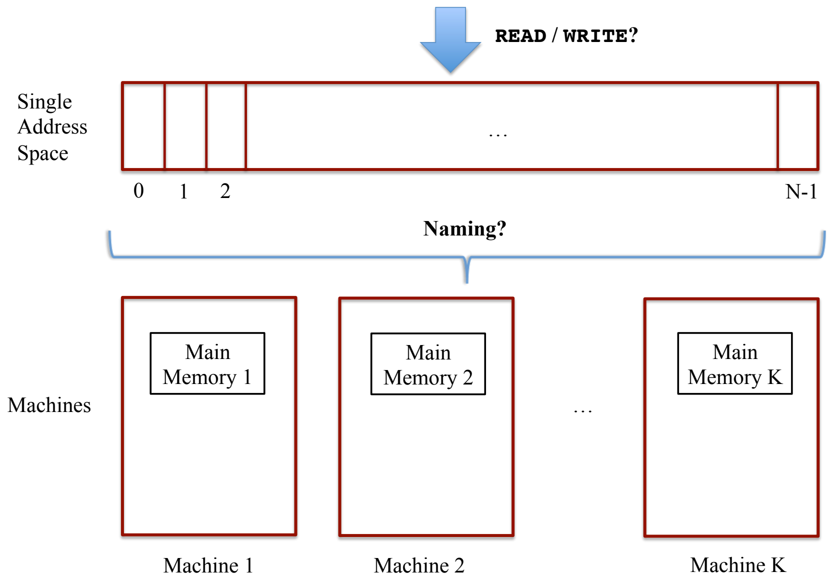
Do not get hung up in a single question. It is best to make an effort on every question and write down all your solutions than to get a perfect solution for only one or two of the questions.

Note that all homework assignments have to be submitted via Absalon in electronic format. It is your responsibility to make sure in time that the upload of your files succeeds. While it is allowed to submit scanned PDF files of your homework assignments, we strongly suggest composing the assignment using a text editor or LaTeX and creating a PDF file for submission. Email or paper submissions will not be accepted.

## Exercises

**Question 1 (Fundamental Abstractions).** Design a memory abstraction that exposes the whole aggregated memory of a cluster of machines as a single address space. You do not need to consider redundancy to mask failures. However, you need to consider that individual machines can fail and have your abstraction handle this in a clean way (i.e., no segmentation fault when a machine is down ☺).

The memory abstraction you should design is schematically shown in the figure below:



To phrase your solution, address the following:

(a) Explain at a high level what technique you would use to map addresses at the single address space to address in each individual machine. Are there any centralized components in your design? How scalable is it as we increase the number of machines? (2 paragraphs)

(b) Show the pseudocode and API of the `READ` and `WRITE` functions of your memory abstraction. Note that your pseudocode should show both translation of addresses (naming) as well as communicate with individual machines to obtain the actual data. (for each function: API + 1-2 sentence description of API + pseudocode)

(c) Are `READ`/`WRITE` operations against regular main memory atomic? Should the `READ`/`WRITE` operations against your memory abstraction be atomic? Discuss why or why not. If you believe they should be atomic, argue how you would achieve that property when multiple clients are accessing your abstraction concurrently. If you believe that they should not be atomic, describe what abstraction you would need to provide to clients to implement atomicity when they need it. (1 paragraph)

**Question 2 (Hardware Trends).** Please evaluate the following three storage technologies (hard disk, solid state disk, main memory) along the following dimensions:

1. Access time – separate out read and write
2. Capacity of an average unit
3. Cost per storage unit
4. Reliability (explain how you quantify reliability)
5. Power consumption

If you use the Web to obtain any numbers or information, remember to cite your sources. Report your results according to the following:

(a) Create a table with the three technologies as columns and the dimensions and their value ranges as rows. Then state briefly your assumptions and sources. (1 paragraph + table)

(b) To implement the scalable distributed memory abstraction you designed in Question 1, it may make sense to use one or many of the technologies above, instead of only main memory. Explain what factors would affect your choice of technologies when designing your system. (1 paragraph)

**Question 3 (Performance).** Regarding techniques to improve performance of a computer system, answer the following questions:

(a) How does concurrency influence latency in a computer system? Is its influence always positive, always negative, or is there a trade-off? Explain. (2 paragraphs)
(b) Explain the difference between dallying and batching. Provide one example of each. (2 paragraphs)
(c) Is caching an example of a fast path optimization? Explain why or why not. (1 paragraph)

**Question 4 (Experimental Design).** [Lilja, partial] Devise an experiment to determine the following performance metrics for a computer system:

(a) The effective memory bandwidth between the processor and the data cache if all memory references are cache hits. Make sure to describe in your setup how you manage to ensure that all references measured are cache hits and why your setup achieves that.
(b) The effective memory bandwidth if all memory references are cache misses. Make sure to describe in your setup how you manage to ensure that all references measured are cache misses and why your setup achieves that.

## Programming Task

In this programming task, you will implement the first version of `KeyValueBase`, a key-value store that we will be refining over the course assignments. By building `KeyValueBase`, we will learn about and experiment with abstractions, techniques for performance, and fault tolerance.

The first version of `KeyValueBase` is single-node and non-durable. However, it already allows its users to manage some datasets that exceed main memory. `KeyValueBase` achieves the latter by operating over a memory management abstraction: a memory-mapped file. The interface of `KeyValueBase` abstracts common operations offered by key-value stores operating over a sorted key space.

### Interface

`KeyValueBase` operates on `Key` and `Value` instances. Keys are taken from a totally ordered domain and are unique in the system. Values are arbitrary other than being serializable. `KeyValueBase` exposes to its users the following service **interface**:

1) `init(String serverFileName)`: This function gets as input a path name to a file stored in the server containing partial key-value mappings and initializes the key-value store. The format of the file is simply one partial key-value mapping per line, with key and value separated by whitespace. A key may appear multiple times in the file, and the full value mapped to that key corresponds to the concatenation of the partial values on each line. For simplicity, you may assume the file is given to you grouped by key. Examples of input files are files representing users in Twitter and their followers, or articles in Wikipedia and the articles they link to. For example, a file could contain the following partial mappings:

```
24    25
25    26
25    27
26    24
26    28
```

The `init` function can only be called once when the service is started. Calling any other function on the service during initialization or calling `init` more than once should result in appropriate exceptions. Calling any other function before `init` is called also results in an appropriate exception.

2) `read(Key k) → Value`: This function returns the value mapped to a given key. The function guarantees atomicity of single key reads, i.e., no other concurrent operation may corrupt the value returned. If the key is not mapped to any value in `KeyValueBase`, then the function raises an appropriate exception. If the data in the example above were loaded in KeyValueBase, then a call to `read("25")` would return `"26 27"`.

3) `insert(Key k, Value v)`: This function atomically inserts a new key-value mapping into `KeyValueBase`. If the key already exists, then the function raises an appropriate exception. Following up on the example above, a call to `insert("27", "29 30 31 32")` would create a corresponding entry for key `"27"` in the system.

4) `update(Key k, Value newV)`: This function atomically updates a key-value mapping in KeyValueBase. As with `read`, if the key is not mapped to any value, then the function raises an appropriate exception. In the example above, if a single client interacts with `KeyValueBase` and calls `update("24", "23 27")`, then a following call to `read("24")` should return `"23 27"`. Note that if there are concurrent clients, other updates to key `"24"` may happen before the `read`, and a different value may then be returned.

5) `delete(Key k)`: This function atomically deletes a key and all its associated information from `KeyValueBase`. If the key is not mapped to any value, then the function raises an appropriate exception.

6) `scan(Key begin, Key end, Predicate p) → List<Value>`: The function scans the key-value store starting with key `begin` and ending the scan on key `end`, all inclusive. All values that qualify predicate `p` are added to the return list. Scan is not an atomic operation, i.e., the list of values returned is not taken from a consistent snapshot. If `begin` is greater than `end`, then the function raises appropriate exceptions.

This interface is inspired by the database backend API of the Yahoo! Cloud Serving Benchmark (YCSB).[1] **Your service must always respect the above interface**.

## Implementation

`KeyValueBase` is exposed to its clients via an RPC mechanism and its implementation must be thread-safe. In addition, it must enforce the atomicity constraints of the API. You should implement this service

[1] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears: Benchmarking cloud serving systems with YCSB. SoCC 2010: 143-154. Available at: http://research.yahoo.com/node/3202.

either in Java or C#. Make sure to use the latest version of Oracle's Java VM or of Microsoft .NET for this programming task. As an RPC mechanism, employ JAX-WS for Java or WCF for C#, and expose your implementation as a web service.

The implementation must be organized into two internal components: the `Index` and the `Store`. The `Index` encapsulates a sorted mapping of keys to file positions in the Store. For simplicity, the `Index` fully resides in main memory. The `Index` provides internal operations to get a position from a key, insert and remove key-position mappings, as well as mapping a range of keys to a list of positions in the `Store`. At its API, the `Index` offers values to its callers, which are internally read from the `Store`. The `Store` itself contains the actual serialized values, which may exceed main memory in aggregate size. The `Store` deals with the latter by encapsulating a memory-mapped file, and providing simple read / write functionality by position. The use of memory-mapped files provides us with efficient I/O as well as buffer management implemented by the OS.

In order to get you started, we provide interfaces corresponding to the service above and skeletons for the main classes in the service implementation.

***Question 1:*** *Which RPC semantics are appropriate for the* `KeyValueBase` *interface? What RPC semantics will you adopt in your implementation? Justify your choice. (1 paragraph)*

***Question 2:*** *Explain the main elements of your implementation of the first version of* `KeyValueBase`*. In particular, describe the mechanisms you used to enforce atomicity of operations (e.g., locks or monitors) and how you used them. In addition, describe how you implemented free-space management. (2 paragraphs)*

***Question 3:*** *Describe how you tested your implementation and ensured that multi-threaded executions produced correct results. (1 paragraph)*

**Experiments**

Once you have your first implementation of `KeyValueBase` ready, you should experimentally evaluate its performance and quantify the impact of your design decisions. The two main performance aspects we will focus on are throughput and latency.

***Question 4:*** *What are the main parameters that influence throughput and latency in the first version of* `KeyValueBase` *(e.g., number of clients, hardware characteristics, size of dataset managed, mix of operations)? Explain why. (2 sentences per parameter)*

***Question 5:*** *Devise a simple back-of-the-envelope analytical model of how the number of clients affects the performance of your service (throughput and latency), given all other parameters are fixed. What behavior would you expect from your service as the number of clients increases? Can you estimate how many clients would be necessary to elicit the behavior you predict? (2 paragraph description of model + 1 paragraph description of prediction)*

As an experiment to quantify the performance of your service, we ask you to test it with real data. Load and query the actual Twitter social graph available from Kwak et al. at http://an.kaist.ac.kr/traces/WWW2010.html. We realize that you may lack private computational resources to use the Twitter dataset. You may optionally use Microsoft Azure / Amazon EC2 or instead select a smaller dataset that fits your computational capacity. If you do this, please select one of the graph datasets available from the Stanford Large Network Dataset Collection (SNAP, available at http://snap.stanford.edu/data/). Make sure to state explicitly in your solution which dataset you used and justify your choice of dataset.

***Question 6:*** *Describe the setup of your experiment with real data. Document all factors that affect the performance of your service. Make every client of your service call only* `read` *operations, with a distribution of keys drawn from a Zipf distribution.*[2] *Carry out the experiment and report your results with two graphs: one with throughput in the y-axis and one with latency in the y-axis. In the x-axis, plot the number of clients used with the service.*

***Question 7:*** *Explain the effects you observe regarding throughput and latency of your service.*

***Question 8:*** *Given your response to Question 4, what other experiments would you use to quantify the performance of* `KeyValueBase`*? Justify your choices. (2 paragraphs) Note: you do not have to execute these experiments, only describe which you would perform and explain why.*

**Extensions**

`KeyValueBase` is a success with developers and they wish to have additional functionality from the system. You agreed to add the following two functions to `KeyValueBase`'s API:

7) `atomicScan(Key begin, Key end, Predicate p) → List<Value>`: This function is an atomic version of `scan`. Except for the requirement of atomicity, all the functionality of `scan` is unchanged.

8) `bulkPut(List<KeyValuePair> mappings)`: This function adds to `KeyValueBase` all key-value mappings given as a parameter. If a key is not present, the key is added; otherwise, the key is updated. This function is required to be atomic, i.e., readers will either see all mappings updated or none updated.

***Question 9****: Briefly describe your implementation of these two functions and how you ensured atomicity. (2 paragraphs)*

[2] Refer to the following paper for information on how to generate different distributions of accesses: Gray et al. Quickly Generating Billion-Record Synthetic Databases. SIGMOD 1994: 243 - 252. Tech report available at: http://research.microsoft.com/~gray/papers/SyntheticDataGen.doc.