Scalable Software Architectures

System Type: Computing Jobs

**1. Reason about which the most important quality attributes might be for the type of system**

All the attributes corresponding to the functional suitability, like functional completeness, correctness and appropriateness should be fulfilled as it is very important that the tasks are being performed in the correct order and the solution is the solution expected.  
Performance Efficiency on the other hand is also important as a huge amount of data have to be processed and therefore parallelization can help to speed up the process, but still ensure that the data is being processed in the correct order.  
Like with all other applications it is important that the result is correct, etc.. But as this is important for all applications, I won’t talk about these in later parts of the assignment.

**2. Reason about a software architecture that can be used to implement the type of system in a scalable way.**

For this application it is important to have a manager who initializes and manages the single agents used for the calculations. The manager gets the task to do assigned and if more working steps are needed to get to the final result, in which order they have to be performed. Then the manager divides the task into subtasks, initializes agents who should run the calculations and send each of them data and the task to perform. As soon as an agent finished his job, he’ll send the result back to the manager and the manager merges the single results into the complete whole result. If after a chain of performances has to be performed, he then sends out subtasks again and waits for the response. It is also possible to just divide the data and let all agents run all the steps without sending back the result after each calculation. But as agents can fail it is better to have a backup of the data at the manager. It can be said, that a MapReduce approach can be used, as this is what I tried to explain. Using this approach, the final result can be achieved using multiple agents in a fast and efficient way.

**3. Reason about how to also ensure Availability and Reliability in the type of system.**

The single point of failure here is the manager who controls all the work to do. If he fails, the whole system breaks down. Normally, the manager should survive, however. If an agent breaks down its not a real problem as the manager can launch a new one who starts the calculation again. The manager controls all the work done and ensures that all tasks are performed in the correct order and makes the final step of the MapReduce procedure.

**4. Reason about how to ensure efficient, scalable, and sufficiently reliable storage of data.**

Here again the MapReduce comes in. The manager has all the data saved or knows the location where the data is saved. He splits the data into chunks that is then send or allocated to a single agent who performs a task on it. The agent receives the data performs his actions and sends the data back to the manager who then again saves the data such that the result gets not lost. It could be useful to have the data of the last iteration and the current one saved as if some problems occur the procedure doesn’t have to start from start again.

System Type: Computing Stream

**1. Reason about which the most important quality attributes might be for the type of system**

Here the same attributes are important again as for the computing jobs.

**2. Reason about a software architecture that can be used to implement the type of system in a scalable way.**

I would also use the MapReduce approach here again. The manager needs to know if the single chunks of data that arrive are dependent of each other or not and if the tasks to be performed should be executed in sequence or if it makes no difference in which order to perform them as they are independent of each other. Let’s assume that the small chunks are not dependent on each other, such that we only have the second problem to resolve. The manager receives the single chunks from time to time. Depending on the size of them, he has to divide them further or assigns the received data to one agent who then performs the data. If the tasks are dependent, I would suggest that one agent performs all the work after each other and always sends the manager its status and result as soon as one task has finished, such that the already computated data gets not lost. If the tasks however are not dependent on each other, the manager could initialize for each data chunk as many agents as tasks have to be performed and each agent performs its task on its own and as soon as it finishes sends the result back to the manager who saves and presents the result in the end. So, I would again use a MapReduce approach where the manager assigns agents tasks to do. If too much data arrives and one manager is not enough, it could also be possible to use one master manager, multiple manager that get controlled by the master manager and then agents that are initialized and controlled by the normal managers. But this really depends on the kind of data and tasks to be performed.

**3. Reason about how to also ensure Availability and Reliability in the type of system.**

Actually, the same as for the computing Jobs type.

**4. Reason about how to ensure efficient, scalable, and sufficiently reliable storage of data.**

Also, the same as for the computing Jobs type.

System Type: User Silos

**1. Reason about which the most important quality attributes might be for the type of system**

As now User are using an application directly it is important that the application is useable and easy to use for the user. All the points mentioned under Usability are important: Appropriateness, Recognizability, Learnability, Operability, User error protection, User interface aesthetics and accessibility. Also, other attributes like availability and Integrity are very important such that the user is satisfied and gets in the end the result he is looking for.

**2. Reason about a software architecture that can be used to implement the type of system in a scalable way.**

Here we need an application server that handles all the traffic and performs all the tasks. If one single application server can’t handle all the traffic, another one should be launched and then a load balancer is also needed that sends the request to one of the application servers. It is important that one users requests always are directed to the same application server such that the integrity holds. Also, a database is needed, that has all of the data needed for the application saved. The architecture needed here is a simple common architecture used for web applications.

**3. Reason about how to also ensure Availability and Reliability in the type of system.**

It has to be ensured that at least one application server and the database is available all the time. As soon as the utilization exceeds a specified percentage of the maximal utilization of a server, a new one should be launched. The same for database slaves, if working with these, such that all the requests can be handled accordingly.

**4. Reason about how to ensure efficient, scalable, and sufficiently reliable storage of data.**

The persistent data should be saved on a database master that handles all the writes and multiple database slaves should process the requests of the users. Also, some backups of the data should be done in regular intervals, such that the data is not being lost should the database master break down or at least not a lot of data gets lost.

System Type: User Silos with Post-Processing

**1. Reason about which the most important quality attributes might be for the type of system**

All of the above mentioned, but also functional completeness, such that really all of the tasks requested are getting performed.

**2. Reason about a software architecture that can be used to implement the type of system in a scalable way.**

Here we again need an application server that handles the traffic and performs the tasks. But as now a user can request multiple tasks, it is important that the application server saves the requests somewhere and then starts the processing of these. As it is not important to send the results back again immediately. Therefore, the tasks can be done one after another and it is not required to perform the tasks in parallel. It should be however ensured that all requested tasks in the end get performed. Here the database is more important such that no request gets lost. As soon as some computing capacity is free, a new task can be started and afterwards the answer gets send to its receiver. However, it has to be ensured, that enough applications servers are online in order to receive all user requests. Servers can be added to process them faster but that is not directly needed and not most important.

**3. Reason about how to also ensure Availability and Reliability in the type of system.**

As said in section 2. Enough application server need to be online to receive all user requests and the database has to be reliable and duplicated such that all requests are being saved properly. Then as soon as some computing power is free requests are being processed. Therefore, the requests can probably be processed in the night- or any-time frame almost no user is online and sends new requests, such that the application server online can be utilized in the best possible way.

**4. Reason about how to ensure efficient, scalable, and sufficiently reliable storage of data.**

Have duplicated databases or make backups quite regularly such that all the data is persistent. Also like for the User Silos with database masters and slaves.

System Type: Networks of Users

**1. Reason about which the most important quality attributes might be for the type of system**

Same as for the User-Silos. But also, Co-existence and Interoperability as now users can see and add/change the informations the others can see in almost real-time. Also, Security attributes like accountability and authenticity are needed as the system should always know who added or changed data.

**2. Reason about a software architecture that can be used to implement the type of system in a scalable way.**

It looks the same as above. Only the application server should ensure that newly added or changed data is being send back to users who are currently on the webpage changed. Therefore, the application needs to have knowledge about all users currently online and on what page they currently are. Multiple database masters and slaves are needed such that the data integrity is always secured. Backups should be done in times less users are online or enough database masters should be up such that it is not a big deal if one of them gets offline for backup for a short duration.

**3. Reason about how to also ensure Availability and Reliability in the type of system.**

It is important that enough application and database servers are online at the same time such that all requests can be handled.

**4. Reason about how to ensure efficient, scalable, and sufficiently reliable storage of data.**

Same as for the User Silos. However, with multiple users (millions+), the databases could also be divided however, such that if one database should break down, other requests can still be answered. Technique that could be used for this is sharding.