# Coursework COMSM0072 CCBD

Version: 09.11.2021 v1.1

Changes:

10.11.2021 v1.1 – internal review updates

09.11.2021 v1.0 – initial

## Summary

In this coursework you are being asked to write a report documenting the construction of an elastically scalable and fault-tolerant big data cloud system that performs a parallelisable task on streaming data. In addition to i) designing, ii) constructing and iii) measuring the performance of the distributed application, we would like you to experiment with it, and show that it iv) scales elastically, and by introducing failures, v) demonstrate that the system is fault-tolerant. We would like you to write a report in the LNCS format.

**Weighting**: This assessment is worth 100% of your total unit 15 credits.

**Set**: **9am Monday 22 November (Week 9)**

**Due**: **Friday 10 December (Week 11) at 1pm; but aim to submit by 12pm**

**Submission**:

Via on the coursework unit (not taught unit) blackboard page under "Assessment, Submission and Feedback" submit:

- one pdf containing an 8-page report in LNCS format (10% penalty for every partial page over) including in the appendix
- a link to the source code in github repository(s) in the github organisation ccdb-uob including deployment instructions of your application and a tag cw_ccbd_2021.
- A zip/tarball archive of all source code required to run your application and perform the analysis.

Expected workload: 8 hours a day, 5 days a week like in a job – not 24/7 for three weeks solid.

Available support during weeks 9–11: Friday 9 am Q&A and office hours, bookable via https://outlook.office365.com/owa/calendar/OfficeHours@bristol.ac.uk/bookings/

In this document we provide a detailed explanation of the task, and the different ways you could go about solving it and the approach to marking. A marking scheme can be found later in this document.

## A Fault Tolerant Cloud Application

In this coursework we ask you to design and construct a cloud application that performs a task of your choice on stream data; in other words, on event data that arrives at fast pace and requires processing

close to event arrival time. We would like you to demonstrate that the application provides an important property of cloud systems – fault tolerance. This system then needs to be empirically evaluated in its base performance (when no faults occur) and when you introduce faults in various system parts. The description of the system and the evaluation of its performance will need to be documented in a report in the LNCS format.

You have complete freedom over your choice of stream data, and task. However, we provide an example with code as a basic solution that has a variety of extension points. Together, this coursework offers every student the ability to pass the coursework by applying the concepts they have been taught plus some self-study. Higher achieving students can demonstrate their abilities with more advanced concepts that require a greater degree of self-study.

## Task

You are free to choose the stream data and task that your application carries out. The use case that the task satisfies should be motivated within your report. In this assessment context, the task implementation can be schematic and simplify or neglect aspects that an implementation for a real-world deployment would require. Please consider some of such limitations in your report.

Whatever task is chosen, it needs to motivate the scalable streaming architecture of your cloud application.

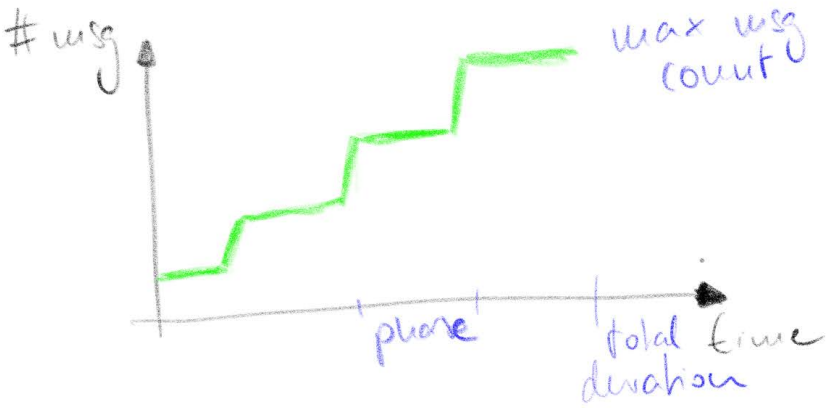
When it comes to implementing the task please note that the purpose of this coursework is to allow you to demonstrate learning of Cloud Computing and Big Data methods. That means that sophistication of the jobs that the worker nodes carry out that falls outside of cloud computing and big data; and inside AI or ML will not be rewarded.

### *Example – Credit Card Transaction Validation*

To combat credit card fraud banks are performing checks on each transaction and can instantaneously fail a payment request if deemed necessary. Due to the real-time nature of payment transactions, these checks cannot be performed in batch, thus constituting a streaming application. The large number and increasing sophistication of the checks, requires big data streaming architectures.

In such a streaming application, credit card transaction requests enter the system on one side of the architecture into a work request queue. A set of worker nodes processes individual transaction requests from the queue, carries out a classification (allow/prevent payment) and feeds the result back.

We have created a simple event generator at https://github.com/ccdb-uob/stream_event_generator. In its current implementation it generates event data (at the moment a static IP address 123.456.789.100) and sends it to a message queue. The number of events per time interval increases in configurable steps as you can see below.




If you chose this application scaffold, there are still some parts you need to implement yourself. Firstly, you have to implement a worker task that processes these messages. One of the most important checks for credit card fraud is based on the location of the IP address associated to a transaction. Accordingly, you could chose to build a rudimentary worker that just looks up the IP address in a geolocation database for IP addresses -- for example, in the free Maxmind DB (https://dev.maxmind.com/geoip/geolite2-free-geolocation-data?lang=en).

You could implement this in a variety of ways. The choice will affect the runtime per worker task.

Without any further info, your worker might decide that IP addresses inside the UK are valid, and anything outside of the UK is invalid. By default, this would result in only one type of response for the IP address 123.456.789.100. However, that would be acceptable as a basic application. A more sophisticated application might seed a DB with transactions and then carry out some actual proximity checks. Such worker tasks would increase its complexity and potentially also require a DB with potential further impact on the runtime of worker tasks.

By balancing the complexity of the worker task and calibrating the maximum message count, number of ramp up phases and total duration you can tune your event rate to the right value, so that you allow your system to demonstrate its ability to dynamic scale and be fault tolerant; while staying within the limits that educational cloud provider offers provide. Note again, that sophistication of the worker job by carrying out ML/AI model building will not be rewarded. What would be rewarded, is the development of scalable database architectures to support your worker jobs.

Once properly tuned, you can analyse the performance of the running system in a variety of ways.

The above is a basic streaming application. There are many other streaming tasks out there. You are free to chose your own, but do make sure you provide some motivation for your choice in the report.

## Choice of Cloud

Your application needs to run in your own application cluster in the cloud.

You can freely chose what cloud to implement the solution on. AWS Academy is a solid choice, given that you have $100 of credits available. If it looks like you happen to exhaust those, please get in touch with us early enough that we can get a top-up arranged.

Something you need to be aware of are resource limits with the educational programs. AWS Academy has these as well. Check out these limits so you don't get surprised.

## Application Architecture

You are free to compose your application as most appropriate for the task. We expect that you will build a system from decoupled components in some way. The greater the degree of components that are outsourced to the cloud provider the easier it might be, but at the expense that you have less opportunity to demonstrate your ability to build scalable applications yourself. So, there is a trade-off.

One of the easiest solutions might be a system using a task queue with worker nodes on top of Elastic Beanstalk. If you choose a PaaS system such as Beanstalk the management of the cluster is outsourced to the cloud provider making it much easier for you to focus on the compute task. For instance, you would scale your cluster up and down as a group without addressing specific instances. Because of the delegation of management tasks, if you do decide to use PaaS you'll miss the opportunity to demonstrate your ability to apply many of the concepts that we discovered in this unit, from use of VMs all the way to synchronisation in distributed systems. Thus, you will miss out on some marks awarded for this higher challenge.

One other decision is where your application controller is located – either on the cloud or on your own computer. At a minimum you can use tools such fabric (see activity to week 5) to deploy your cluster from your laptop and have the controller monitor the state of your cluster from your local computer. If you do run your application from your local computer with AWS Academy you will have to take into account that your auth keys are only valid for a limited session duration.

Once your application is running you should take performance measures. This can include aspects such as the degree of variability of throughput or network latency between nodes ("are all nodes equally fast"). As you monitor the performance of the cluster your controller has to bring failed instances back online. This will take some time. You can develop a regression model of progress degradation over instance failure rates (you can introduce your own delays to make this more pronounced).

## Fault Injection

In order to demonstrate fault tolerance you will have to disrupt some components of your architecture. Netflix has coined the term "chaos testing" to refer to randomly turning off nodes in an application.

How you go about doing this is up to you. As with the other aspects there are levels of complexity to this as well. In the most basic approach you might ask your controller to randomly terminate worker nodes (that then need to be rescheduled after some delay, maybe).

## Github

Your entire source code must be included in one or more repository in the ccdb-uob organisation. You need to provide specific deployment instructions in the README.md file in the repository.

The repository must be private.

You also must create a tag of the repository with a timestamp of or before the deadline (see above), labeled cw_ccbd_2020.

You also need to include the source code of your cloud application, any local controller code and code used in the analysis of the results.

## Report

Your paper should be formatted according to the Springer Lecture Notes in Computer Science (LNCS) conference-proceedings format, details of which are available here: http://www.springer.com/computer/lncs?SGWID=0-164-6-793341-0

Your paper should be no longer than eight pages in LNCS format, including all figures, references, and any segments of code you include to explain your application. You can write 8 pages of text and graphs and tables explaining your work. The appendix is not included in the page count. It should include the link to your github repository on github.com/ccbd-uob.

Every partial page over the 8 page limit will incur a 10% mark penalty.

### Writing Tips

Here we list miscellaneous resources that may be useful across the degree; they're offered in no particular order.

- Dave Cliff's Accessibility score: Low Click to improve TipsOnWriting is a sequence of tips on how to avoid some common problems when writing
- The classic William Strunk's *The Elements of Style* in UoB Library: https://bris.on.worldcat.org/oclc/854969873.
- Steven Pinker's lecture on writing, at the Royal Institution.
- Larry McEnerney from the University of Chicago gives very good advice on postgraduate-level technical writing in this lecture.
- How to Speak. In this video MIT's Patrick Winston gives a one-hour tutorial in how to speak. It is well worth watching.
- Professional proof-readers and copy-editors can help improve your writing. Examples of specialists in academic reading/editing are: Typewright, Margaret Towson, and TheFilthyComma, although lots of other providers are available.

## Marking

**Group Work**

You have already told us what groups you would like to work in. If this changes in any way you must tell us immediately. No changes to group composition can be accepted after Monday 29th of November, noon.

If any group member experiences external difficulties that affect his ability to contribute to the project, the group must inform us immediately in writing.

If one of your group members is subject to circumstances that qualify as EC, the regular UoB EC process will apply. Other group members will be assessed, taking into account the reduced group size, discount in proportion to the time available to the assessment submission deadline.

**Marking Bands**

Below marking bands with maximum possible mark range achievable given approximate scope of work.

| Band | Criteria |
|---|---|
| 80% | Outstanding report. Extensive exploration and analysis demonstrating deep understanding and reading outside of the lectures. Report worthy of publishing in good conference or journal. System implementation shows novel approaches. |
| 70 - 80% | All of the criteria for 60-70, plus lifting the quality of the report to excellent quality, including: Wide set of distributed system components are used; Some novelty in design choices; The system's ability to withstand failures has been robustly and comprehensively demonstrated. Advanced notation and architectural clarity. |
| 60 - 70% | **Code** demonstrates all learning outcomes, for example self-managed synchronisation of distributed systems components, automatic/programmatic scalability, realistic worker tasks.<br><br>**Report** clearly describes the architecture and how it supports scalability, using the notation from the lectures, and includes an appropriate empirical evaluation of the scalability behaviour. The report starts with an abstract containing a clear motivation of the project and a summary of the results, and finishes with a interpretation and a closing summary. Key terms are defined in a way accessible to a reader with a broad general knowledge of cloud computing; specific technologies used are explained. The evaluation of the project is realistic and honest, and discusses technological choices made.<br><br>**Presentation** of the report includes clear and economical writing of the correct length. Correct spelling and high quality phrasing. Diagrams are clearly readable (font size, captions) including when printed in black and white (for example use patterns/stripes as well as colours to distinguish data series) and add value in the form of evidence, summary or interpretation. References are consistent. Key results are clearly presented. |
| 50 - 60% | Satisfactory report, showing attainment of most learning outcomes, with some shortcomings. This could be regarding the level of management of infrastructure (e.g. relying on default PaaS or CaaS setup with manual failure injection), or basic worker implementation, relying on example task, basic statistical analysis and results. |

| | |
|---|---|
| <50% | Report is not at an appropriate standard. Objectives of the assignment have not been (fail) demonstrated. |

**Academic Offences**

Academic offences (including submission of work that is not your own, falsification of data/evidence or the use of materials without appropriate referencing) are all taken very seriously by the University. Suspected offences will be dealt with in accordance with the University's policies and procedures. If an academic offence is suspected in your work, you will be asked to attend an interview with senior members of the school, where you will be given the opportunity to defend your work. The plagiarism panel are able to apply a range of penalties, depending the severity of the offence. These include: requirement to resubmit work, capping of grades and the award of no mark for an element of assessment.

**Extenuating circumstances**

If the completion of your assignment has been significantly disrupted by serious health conditions, personal problems, periods of quarantine, or other similar issues, you may be able to apply for consideration of extenuating circumstances (in accordance with the normal university policy and processes). Students should apply for consideration of extenuating circumstances as soon as possible when the problem occurs.

**Q&A**

- **Level of sophistication of a certain feature**: Polishing a feature will provide you with increasingly diminishing returns in terms of marks. By investing time into making a feature perfect and compromising on other areas (see marking rubric) you might lose marks compared with a more well-rounded piece of work.
- **Can we use S3, Kubernetes, XYZ?** You can use any technology that is reasonable for your system. You will make a choice of technology/solution based on some literature review (includes blog/tech articles on the web) and your own conceptual understanding. What matters is that in your report you include this reasoning and argumentation.
- **Is my task appropriate?** Your task should have some utility to someone that you should illustrate in the introduction/background. More importantly, the task should genuinely benefit from increasing the number of nodes in your cluster. If your task can be solved by a small number of nodes already, then you are potentially depriving yourself of the ability to evaluate your systems performance under failure.