The space time complexity

Problem statement

It is year 2031 and a collective consisting of leading scientists on the earth called TSF (*The space frontiers*) has open sourced the design for a nuclear powered space probe. It is tiny in size, can be 3D printed and assembled, launched with specialized propellants from anywhere and has a really long life span. It makes it an ideal candidate to send it for non critical data collection missions into deep-space. This has given rise to a movement **#mapOurUniverse**. TSF is crowdsourcing the effort to launch millions of probes to help map the entire universe. These probes pair up to create a mesh and each probe is configured to send data packets of their environment and location to a server back on the earth.

You have been set incharge to create a ***critical***, performant and resilient **data store from scratch** that can store and maintain the latest entry sent by each probe at any given time. This is a crucial low level component for building a map of the universe as it records the farthest known edge of the universe at any given time. Several other systems query & act on data collected in this system in real time to adjust probe flight path.

Following are the demands from the system to work as expected

- The system should expose an HTTP endpoint to accept a PUT request containing a payload specified below in the document
- The system should create/update the payload based on a unique probe id. The spec for probe id is specified below in the document
- The system should not lose data, if it undergoes a planned (an explicit process SIGTERM/SIGQUIT) or an unplanned (*due to a failure, SIGKILL*) restart
- The system needs to have only the latest record per probe. Older records can be overwritten safely.
- The system needs to expose an HTTP endpoint to retrieve the latest record by a probe id
- The 99p response time for writing a record to the service should not go beyond 200ms.
- The probes may have unreliable gaps between two subsequent transmissions. This should not impact the availability of their last available record
- The system should also allow data to be queried from the system by a probe id. The contract and nature of data queries are specified in the section below
- The system should identify latest event using eventTransmissionTime and discard older events in case they are received in an incorrect sequence (*an older event after the latest one*)

Probe Id spec

An alphanumeric id. It has to be unique for a probe. A probe can have only one probe id. At Least 3 characters and maximum 100.

Payload spec

Average message size - 5 kb (Min 1 kb to max 20 kb)

Field name	Description	Value format
probeId	A unique id per probe hardwired into the probe sent in each transmission	Spec for probe id specified below
eventId	A unique id per event.	UUID with timestamp built in with millisecond precision
messageType	A message classification. For this scenario it will always be “spaceCartography”	Fixed string
eventReceivedTime	Timestamp when the event information was received. Needs to be populated by the receiving applications/datastores	Milliseconds since epoch
eventTransmissionTime	Timestamp when the event was transmitted from the probe. Injected into the payload by the probe	Milliseconds since epoch
messageData.measureName	Name of a measurement. Total 6 types of measurements exist	Fixed string set
messageData.measureCode	A short code for a measurement. Total 6 types of measurement code exist **SCSED** - The euclidean distance from earth to	Fixed String set

	the probe as per the specifications of Spherical coordinate system **SCSEAA** - The azimuth angle from earth to the probe as per the specifications of Spherical coordinate system **SCSEPA** - The polar angle from earth to the probe as per the specifications of Spherical coordinate system. Measured in parsecs **LER** - Frequency of electromagnetic radiation in the current environment / space **PLSE** - Remaining life span for the probe in number of years accounting current wear and tear **PDL** - Diagnostic logs from the probe	
messageData.measureUnit	Unit of measurement for the specified measurement	Fixed String set
messageData.measureValue	The actual value derived/raw for the measurement	**SCSED** - Floating point **SCSEAA** - Floating point **SCSEPA** - Floating point **LER** - Floating point **PLSE** - Floating point **PDL** - Text

messageData.measureValueDescription	Description of the measurement	Fixed string set
messageData.measureType	Category classification for the measurement. Three categories exist currently. **Positioning** - The position of the probe in space in relation to the position of Earth. **Composition** - The composition of the space around the probe **Probe** - Details about the probe itself	Fixed string set
messageData.componentReading	The raw reading from the underlying component in case the measured value is derived	Floating point

Sample Payload

```
{
  "probeId": "PRB34222422421123",
  "eventId": "7707d6a0-61b5-11ec-9f10-0800200c9a66",
  "messageType": "spaceCartography",
  "eventReceivedTime": <TO BE POPULATED BY DATA STORE AS EPOCH WITH
MILLIS>,
  "eventTransmissionTime": 1640018265951,
  "messageData": [
      {
           "measureName": "Spherical coordinate system - euclidean
      distance",
           "measureCode": "SCSED",
           "measureUnit": "parsecs",
           "measureValue": 5399e5,
           "measureValueDescription": "Euclidean distance from
      earth",
           "measureType": "Positioning",
           "componentReading": 43e23
      },
      {
```

```
      "measureName": "Spherical coordinate system - azimuth
angle",
      "measureCode": "SCSEAA",
      "measureUnit": "degrees",
      "measureValue": 170.42,
      "measureValueDescription": "Azimuth angle from earth",
      "measureType": "Positioning",
      "componentReading": 46e2
},
{
      "measureName": "Spherical coordinate system - polar
angle",
      "measureCode": "SCSEPA",
      "measureUnit": "degrees",
      "measureValue": 30.23,
      "measureValueDescription": "Polar/Inclination angle from
earth",
      "measureType": "Positioning",
      "componentReading": 56e42
},
{
      "measureName": "Localized electromagnetic frequency
reading",
      "measureCode": "LER",
      "measureUnit": "hz",
      "measureValue": 3e5,
      "measureValueDescription": "Electromagnetic frequency
reading",
      "measureType": "Composition",
      "componentReading": 3e15
},
{
      "measureName": "Probe lifespan estimate",
      "measureCode": "PLSE",
      "measureUnit": "Years",
      "measureValue": 2390e2,
      "measureValueDescription": "Number of years left in
probe lifespan",
      "measureType": "Probe",
      "componentReading": 6524e3
},
{
      "measureName": "Probe diagnostic logs",
      "measureCode": "PDL",
      "measureUnit": "Text",
      "measureValue":"some log data from probe",
      "measureValueDescription": "the diagnostic information
from the probe",
          "measureType": "Probe",
          "componentReading": 0.0
     }

  ]
}
```

The payload can be stored in any format as long as it can be constructed back and returned as provided.

Probe data payload contract

REST **PUT** /probe/<probe_id>/event/<event_id>

This endpoint will be used by the probe simulator to submit packets and test the solution throughput. If the probe id doesn't exist then a new record should be created for the probe with respective event/event id.

Probe query contract

REST **GET** /probe/<probe_id>/latest

This endpoint will be used by the probe simulator to test the availability of the data that was posted previously and the time within which it is available

Body

As specified in payload spec above

Evaluation criteria

Criteria	Point allocation formula
99p response time for saving a record in data store < 200 ms	
Each probe will transmit a message containing above payload every 10 seconds	
Points based on peak load factor sustained for 120 minutes Load factor = *Numbers of probes supported within thresholds specified above.*	X *where X = sustained peak probe count
Records written to the data store > 5000 ms should not be lost during a planned/unplanned restart. For durability	$X * \left(4^{\left(1 - \left(\text{<guaranteed-durability-millis>} \frac{\square}{\square} 5000 \right) \right)} \right)$

guarantees less than 5000 ms, there will be bonus points as per the formula specified here. For durability guarantee more than 5000 ms there will be a penalty for the points as demonstrated in the formula here	*where X = *sustained peak probe count with claimed durability duration* For instance A peak probe load of 3000 probes with a guaranteed durability after 100 ms will be awarded 11671 points. A peak probe load of 10000 probes with a guaranteed durability after 4900 ms will be awarded 10281 points. A peak probe load of 30000 probes with a guaranteed durability after 10000 ms will be awarded 7500 points. A peak probe load of 100000 probes with no guaranteed durability will be awarded 0 points.

Guidelines and constraints

- Once the problem statement is released you can start developing a solution. Specific hackathons may be arranged to carve out time to enable people to build solutions, but the time is not restricted to that.
- Any runtime/technology can be used to develop the data store itself. As long as the fundamental storage capability is not pre-built. E.g. using an existing storage engine.
- You cannot use a pre-existing data store (Relation, NOSQL or any other type). The data store should be hand crafted, custom-built and tuned.
- Given the test scripts will expect an HTTP api, either the database can expose an HTTP api directly or it can be fronted by a web service to channel Reads/Writes
- No intermediary caches or storage side cars are allowed, in case the data store is fronted by a web service. This is different from in memory indexes that you may create for the data store.
- Everyone will be provided with the exact same hardware. The machines are likely to be Ubuntu **c6gd.large** *(4 GiB, 1 x 118 NVMe SSD, Up to 10 Gigabit)* machines in AWS Mumbai region. This instance comes with a NVMe SSD attached to the instance. The default capacity of the SSD will be flat 20GB for all submissions. However once an implementation crosses 1 million concurrent probes, for each subsequent million probes an additional 20 GB will be allocated.
- There will be an external script that generates payload with the spec specified in the document. The devised solution should be able to interoperate with the external script
- The cloud machines will be provisioned after the requested intermediary stats have been published from local development. Details on intermediary stats will be available once the competition kicks off.

Evaluation process

- Big O kicks off (Overall duration 8 weeks)
- Each team starts building data store locally on their machines
- Intermediary stats are requested. Teams that publish the stats get the next level of resources at each step.
    - Access to AWS
    - Access to test scripts
    - Access to final testing setup and so on.
- Sessions and hackathons are organized to provide ideas, knowledge, tips/tricks & pizzas
- Each team is able to test on AWS with final scripts and measure their storage performance
- Evaluation setup is created and executed for all qualifying teams
- The top two teams with highest sustained load factor are declared as winners

Hackathon schedule

Date	Event / Location
~~20th Dec 2021~~	~~Kick off event with email requesting registration~~
~~22nd Dec 2021~~	~~Broadcast problem statement to the registered teams~~
~~24th Dec 2021~~	~~Close registrations~~
~~7th Jan 2021~~	~~First hackathon~~
21st Jan 2021	Second hackathon
4th Feb 2021	Third hackathon
18th Feb 2021	Midnight submission close
18th Feb 2021 - 28th Feb 2021	Daily showdowns and showcases (one team at a time)
Annual day	Winner declaration and prize distribution