1 1998 Paper 12 Question 1

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y1998p12q1.pdf

For a large-scale distributed system or application:

(a) Describe alternative approaches to creating unique names. Include a discussion of the information conveyed by a name.

Put the most looked-up or most safety-critical data in the name.

A name is a unique identifier paired with information about the node. The information paired with the node can be arbitrary but should be immutable and generally relevant. For example information about the hardware or software the node is running or a "priority". If we assume a fail-stop model then the time the node was created would also be reasonable. Critically, the information in the nodes name can only be provided by the node which named that node. So if a node named itself then it may not know its time of creation or its own priority.

Structure of the name can make it easier to search up -- ie chu.cam.ac.uk: search uk, then .ac.uk, then cam.ac.uk then cam.ac.uk etc etc

There are a three main ways to generate a unique identifier:

Reduce coordination by having multiple levels of heirarchies

- Delegated by by a naming service such as a router or leader (RAFT).

  This unique identifier can simply be a counter. If given by a "superior" then it should use a unique "superior ID" as a prefix. "Partition the namespace" -- uniqueness without coordination

- A function of something known to be unique. Best practice to delegate blocks rather than "half"

  For example the MAC address or the ID of the user who is working on the node paired with the wall time at which they logged in.

  changes of name?

No information for ie networking

You can only test for equality or look up -- you now need a distributed hash table.

Find is now linear in the total number of names globally -- this is VERY inefficient.

A solution is to have the name as a pair or random number and information

- A random number

  This is not guaranteed to be unique; but can be designed such that the probability of uniqueness is astonishingly high. For most systems, a 64-bit number would be sufficient and could even be paired with the current wall time to bring the probability of uniqueness even higher and stop the probability of ever having a collision in a given second increasing.

  Random names can be really long -- if you name a small object then the size of the name can be longer than the size of the object!

  Random bits are good when you need to name things at a very high frequency.

  For example, if we used a random 128-bit integer paired with the wall time (ms-precision) then the probability of the Internet having a name collision in the next million years is $\approx 1.4 \times 10^{-14}$. This assumes that the seed to the random number generator is *not* the wall-time.

(b) Discuss name to location binding under the headings

The only way to be more available than the lookup service is to have information about the server in the name itself.

(i) availability of the binding service

design so avail doesn't matter?

Binding service is the server which maps name to node/website/...

To get high availability of the binding service, the binding service should be local. In this case either a random number or a function of an existing unique identifier is preferable. If names are given by a leader then the node may have to wait to be given a name.

e.g. a cache?

If names change then caches become out of date.

(ii) consistency of the naming data

To ensure data put in the name is consistent, the data should all come from teh same source: a coordinator. In this case it is preferable to use strategy one – where a leader or superior delegates names.

Much easier to build the naming layer if you assume data won't be consistent.

(iii) mobility of named objects

Mobility is hard

If a leader delegates a name, then it may not be unique on other networks. This is therefore unusable. Additionally, if network-specific information (such as time of joining or priority) is included in the name then the named object is not portable. Names generated under strategy 2 or strategy 3 are globally unique.

2 1999 Paper 9 Question 1

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y1999p9q1.pdf

Describe one of the following architectures for supporting distributed programming:

Either Remote Procedure Call

or Object Request Brokerage

RPC is a method of providing location transparency which involves wrapping a function call to another node inside a "stub" which has the same signature and arguments as the function. From the users perspective, they make a local function call to the stub.

Marshalling tries to be hardware and programinig language independent; this is why JSON.

You cannot marshall pointers!

When the stub is called, instead of performing the function, it marshalls the arguments into a transmittable format (ie JSON) then sends them across the network to the node on which the function is to be performed. The target node then unmarshalls the data and performs the function. Dependent on the exact semantics, a response or return value may be sent or the original sender may be polled.

Make a unique ID for the request. Send the request, function and marshalled arguments. Then have a timer. When the timer expires, block the thread and wait for responses.

and how do you handle the error cases?

The server keeps logs so it doesn't re-run functions.

Many emerging applications require timely responses to events. Discuss the following approaches to achieving this:

(a) polling

Polling is the process of regularly checking if an event has happened. In the context of distributed systems, this would mean sending a message to the node on which the RPC was performed every τs. This is a very inefficient strategy and will greatly increase network traffic.

Allows you to tell the difference between the event and the failure of the server which would report the event.

inefficient IF you need a low latency response to a rarely occurring event.

(b) synchronous call back

Clients now have a lot of threads. A LOT of overhead.

In synchronous call-back, the thread which made the RPC blocks until the call-back returns at which point it is immediately woken. This is analagous to a blocking system call in an Operating System.

You have a thread. It makes an RPC to the thread you subscribe to. Then sleep until it returns.

yep, blocking pub-sub

Often implemented by waiting on a semaphore.

publish / subscribe

(c) asynchronous notification

In asynchronous notification, the thread which makes the RPC can continue executing and will receive an asynchronous notification when the RPC completes. This is analagous to an asynchronous IO operation in an Operating System.

Have an event queue. Events in the queue cause a function to be called.

You now only need one handler thread and one worker thread.

much more complex to program
less thread-heavy to performant

If you subscribe to 10 events, then on any line the computer could be in any of 2^10 states. This makes programming very challenging.