INFO8002

Large-Scale Data Systems

Exercise Session #4

Academic year 2021-2022

LIÈGE université

Reminder

REMINDER :

Distributed Hash Tables

- A **distributed hash table** (DHT) is a class of (decentralized?) distributed systems that provide a lookup services similar to a hash table.
    - Every node in a distributed hash table is responsible for a set of keys and their associated values.
    - The **key** is a unique identifier for its associated data value, obtained using a hashing function.
    - The **data values** can be any form of data.

REMINDER :

Distributed Hash Tables

Chord Algorithm: a protocol and algorithm for a peer-to-peer distributed hash table.

<u>REMINDER</u> :

<u>Distributed Hash Tables</u>

<u>**Chord Algorithm**</u>: a protocol and algo... for a ...-to-peer (**Not Really!** ...e.

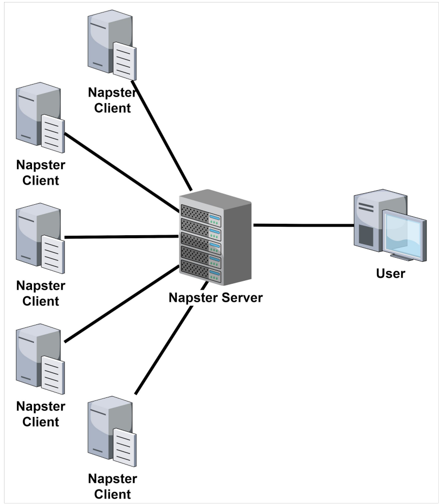
- **Interface**:
  - Support <u>a single operation</u>: **lookup(k)** → Return the ip of the host which hold the data associated to **k.**
- **Properties:**
  - If **k** is stored on the DHT, a process will **eventually** find a node which stores **k**.
  - **Termination**

Help to locate where a resource is!

PROBLEM 1

Content Sharing System

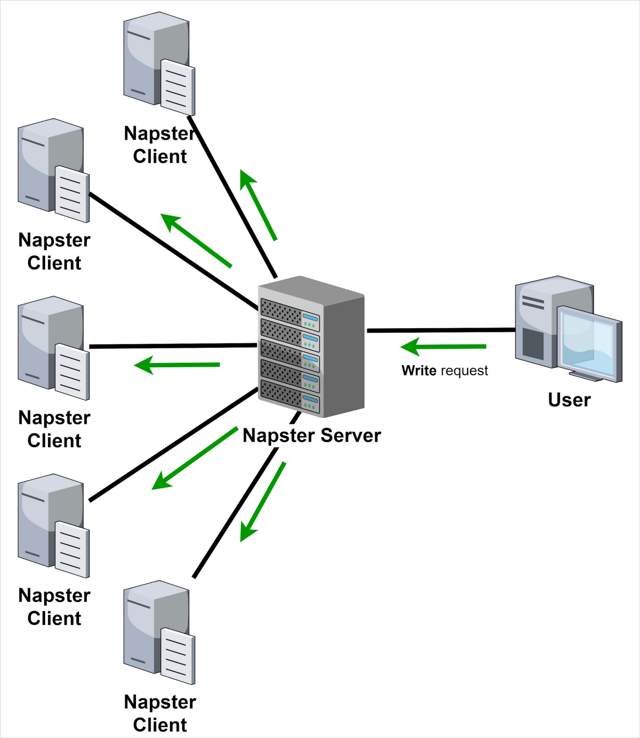
You are responsible for designing a system allowing the **storage** and **distribution** of content.



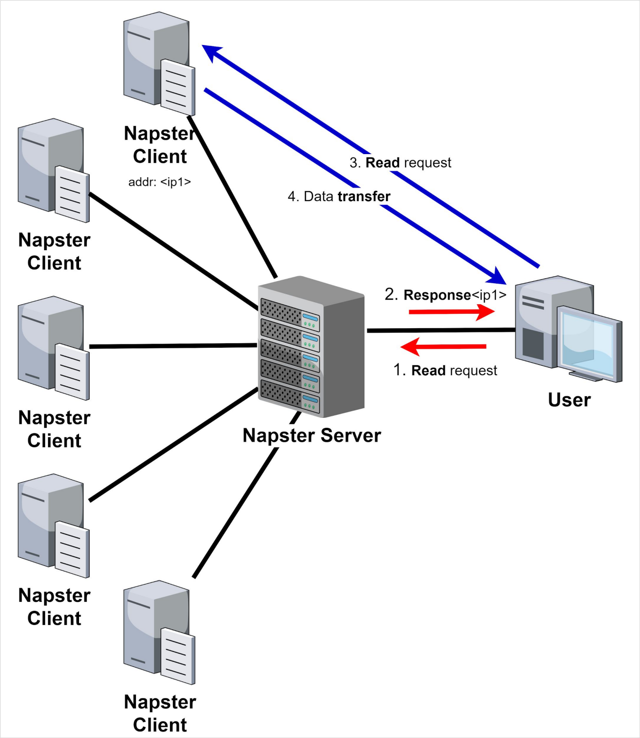
Specify an architecture for this distributed storage system and **provide** a pseudo-implementation using distributed hash tables.

Content Sharing System – 1st Design

Write query



Read query



Content Sharing System – 1st Design

Module Specification:

Module 1: Interface and properties of distributed storage

Module:

Name: *NapsterClientServerRegister*, **instance** *npr*.

Events:

Request: < *npr*, Read | r, m > **:** Invokes a **read** operation on **m** consecutive registers starting on register **r**.

Request: < *npr*, Write | v, r> **:** Invokes a **write** operation with value **v** starting on register **r**.

Indication: < *npr*, ReadReturn | v > **:** Completes a **read** operation with return value **v**.

Indication: < *npr*, WriteReturn> **:** Completes a **write** operation.

Properties:

NP1: Termination.

NP2: Validity.

Content Sharing System – 1^{st} Design

Implementation

Algorithm 3:

Implements:
NapsterClientServerRegister, **instance** *npr*.
Uses:
(1, N)-NewRegularRegister, **instance** *onnrr*.

upon event < *npr*, *Init* > **do**
pendingR **:=** *pendingWr* **:= ∅;**

upon event < *npr*, Write | v, r > **do**
forall v' ∈ v **do**
pendingWr := *pendingWr* ∪ {r + **index**(v)};
forall v' ∈ v **do**
trigger < *onnrr*, Write | v', r + **index**(v)>;

upon event < *npr*, Read | r, m > **do**
ReadRet := $[0]^m$; *offset* := *r*;
for *i* ***in range(m) do*** *pendingR* := *pendingR* ∪ {r + i};
for *i* ***in range(m) do trigger*** < *onrr, Read* | *r + i* >;

upon event < *onrr*, ReadReturn | r, v> **do**
pendingR := pendingR \ {r};
ReadRet[r-offset] := v;
if *pendingR* ⊆ ∅ **then**
trigger < *np*, ReadReturn | *ReadRet* >;

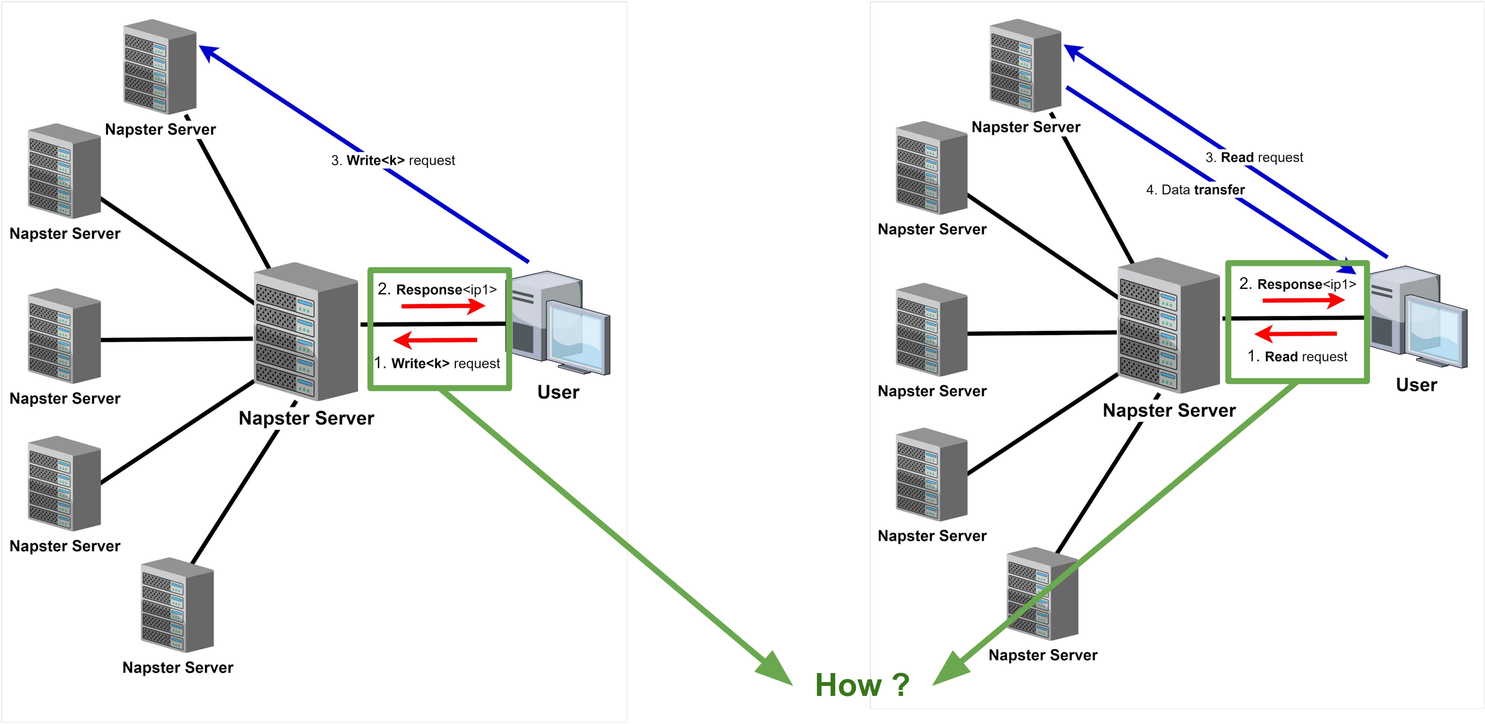
upon event < *onrr*, WriteReturn | r > **do**
pendingWr := pendingWr \ {r};
if *pendingWr* ⊆ ∅ **then**
trigger < *onnrr*, Flush>

upon event < *onnrr*, FlushReturn > **do**
trigger < *np*, WriteReturn >;

Content Sharing System – Updated

Write query

Read query



Content Sharing System – Updated

Module Specification:

Module 1: Interface and properties of distributed storage

Module:

Name: *NapsterServer*, **instance** *np***.**

Events:

Request: < np, Read | k > **:** Invokes a **read** operation of value with key **k**.

Request: < np, Write | k, v > **:** Invokes a **write** operation of value **v** with key **k**.

Indication: < np, ReadReturn | v > **:** Completes a **read** operation with return value **v**.

Indication: < np, WriteReturn> **:** Completes a **write** operation.

Content Sharing System – Updated

Problems:

1. **How can we ensure that operations terminates ?**

   Property n°1: **"Termination"**

   "If a correct process invokes an operation, then the operation eventually completes."

2. **How can we ensure that users receives a coherent response to their read request ?**

   Property n°2: **"Validity"**

   *"A **get** that is not concurrent with a **put** returns the last value written; a **get** that is concurrent with a **put** returns the last value written or the value concurrently written or no value."*

Content Sharing System – Updated

Module Specification:

Module 1: Interface and properties of distributed storage

Module:

Name: *NapsterServer*, **instance** *np***.**

Events:

Request: < np, Read | k > **:** Invokes a **read** operation of value with key **k**.

Request: < np, Write | k, v > **:** Invokes a **write** operation of value **v** with key **k**.

Indication: < np, ReadReturn | v > **:** Completes a **read** operation with return value **v**.

Indication: < np, WriteReturn> **:** Completes a **write** operation.

Properties:

NP1: Termination.

NP2: Validity.

Content Sharing System – Updated

Implementation

Algorithm 1:

Implements:

NapsterServer, **instance** *np*.

Uses:

Chord, **instance** *chord*.

NapsterClientServerRegister, **instance** *npr*.

upon event < *np*, *Init* > **do**

???

upon event < *np*, Write | k, v > **do**

???

upon event < *np*, Read | k > **do**

???

upon event < *chord*, lookupComplete | k, ip, r, m > **do**

???

upon event < *npr*, ReadReturn | v> **do**

???

upon event < *npr*, WriteReturn > **do**

???

Content Sharing System – Updated

Implementation

Algorithm 1:

```
Implements:
    NapsterServer, instance np.
Uses:
    Chord, instance chord.
    NapsterClientServerRegister, instance npr.

upon event < np, Init > do
    pendingR := ∅;
    pendingWr := ∅;

upon event < np, Write | k, v > do
    pendingWr := pendingWr ∪ {k : v};
    trigger < chord, lookup| k >;

upon event < np, Read | k > do
    pendingR := pendingR ∪ {k};
    trigger < chord, lookup| k >;

upon event < chord, lookupComplete | k, ip, r, m > do
    if k ∈ pendingR then
        trigger < npr, ip, Read | r, m >;
        pendingR := pendingR \ {k};
    if k ∈ pendingWr then
        trigger < npr, ip, Write | pendingWr[k], r >;
        pendingWr := pendingWr \ {k};

upon event < npr, ReadReturn | v> do
    trigger < np, ReadReturn | v >;

upon event < npr, WriteReturn > do
    trigger < np, WriteReturn >;
```

Content Sharing System – Updated

Chord vs Kademlia:

- **Kademlia** specifies how values should be stored & retrieved.
  - Resilient to node failures with persistent data storage.
  - Handling of nodes leaving/failing → straightforward (**by design**).
  - Provable properties in 1st paper → **Chord** needed further specifications[1].

- **Performance** comparisons of both algorithms are debatable depending on situations[2].

- **OTHER DESIGNS**:
  - CAN, Pastry, Tapestry → **O(***log(n)***)** lookup (2001),
  - Koorde (2003) → **O(***log(n)/log(log(n))***)** lookup but **complex** to implement**!**

1. Zave, Pamela. "Using lightweight modeling to understand Chord." *ACM SIGCOMM Computer Communication Review* 42.2 (2012): 49-57.
2. Li, Jinyang, et al. "Comparing the performance of distributed hash tables under churn." *International Workshop on Peer-to-Peer Systems*. Springer, Berlin, Heidelberg, 2004.