Data replication

in large-scale distributed systems

Sébastien Monnet

Savoie Mont Blanc university / LISTIC

Who am I ?

- PhD Thesis in IRISA (University of Rennes I)
  - "*Data management for computational grids : a framework for handling fault tolerance and data consistency*"
  - JuxMem prototype
- Post-doc in Italy
- 2007—2016: Associate professor -- LIP6 (Paris 6, now Sorbonne University)
  - HDR : "*Contributions to data replication in large-scale distributed systems*"
- Since Sept. 2016: Professor -- LISTIC (USMB)
- Keywords
  - Distributed systems; systems
  - Data management; fault tolerance; data consistency
  - System virtualization; cloud/edge/fog computing

Data management for large-scale distributed systems

- Motivations
  - Massive data
  - Need for reliable, efficient and consistent data management systems
- Targeted architectures
  - Data center/clouds
  - Fog
- Problems
  - Large scale (high number of nodes, huge volumes of data)
  - Dynamicity (hardware and software)
  - Virtualization (resources fragmentation)

Data replication
in large-scale distributed systems

- Replication : key mechanism
  - Handling multiple copies of the same piece of data

- Three main research axes
  - Fault tolerance (data durability/availability)
  - Access performance
  - Data consistency

Axe 1: Fault tolerance

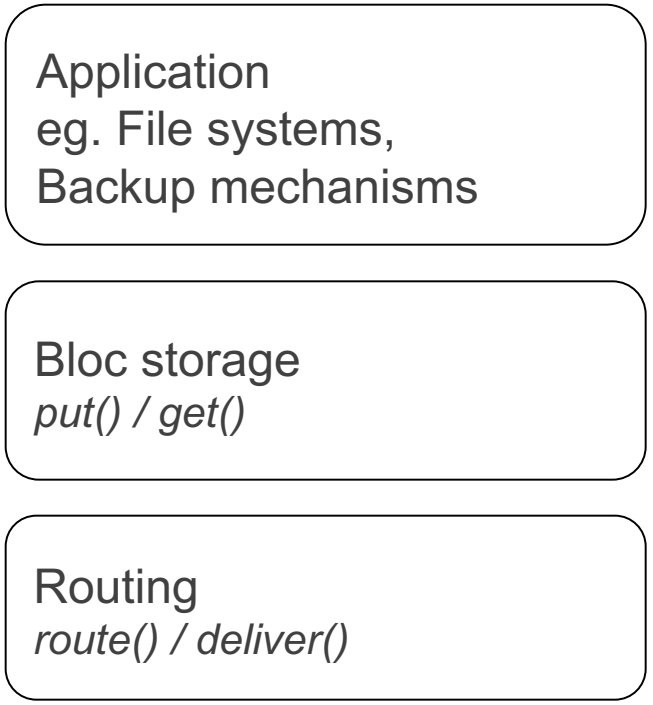
- One piece of data -> multiple copies
  - In case of failure some copies may remain available
- Problems
  - How many copies (replication degree) ?
  - How to place data copies ?
  - How to maintain the replication degree ?
    - When reparation should occur ?

Impact of data placement on data durability

RelaxDHT and SPLAD

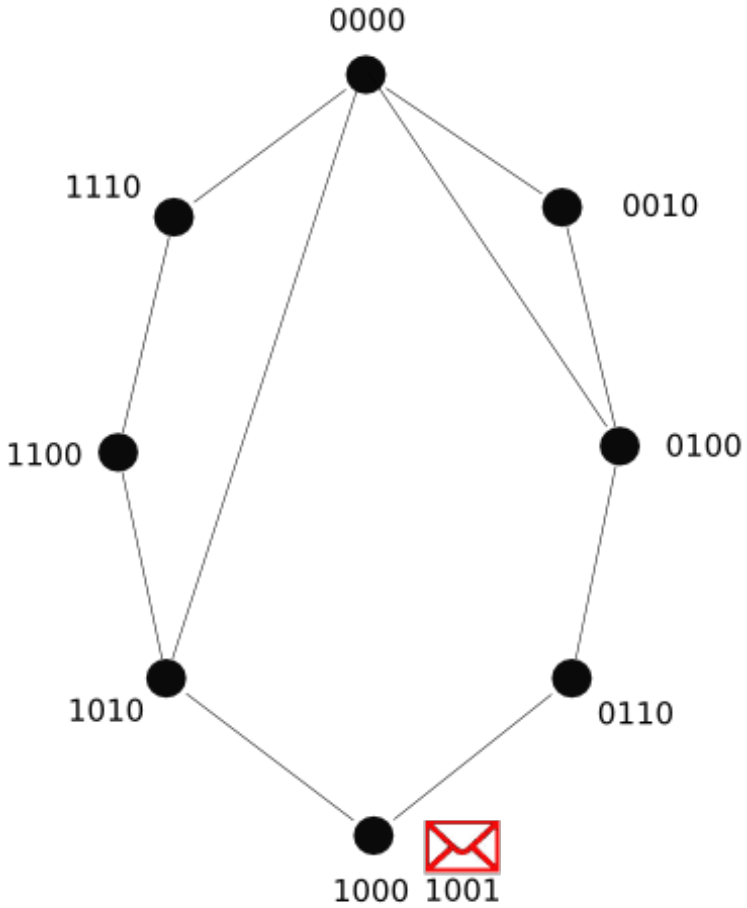
Context : distributed hash tables (DHT)

- Simple API (key/value store)
  - *put/get*
- Efficient localization and routing
- Healing mechanisms (replication degree)
- Many implementations
  - PAST, DHASH, Can, Tapestry, …
- Bad churn tolerance



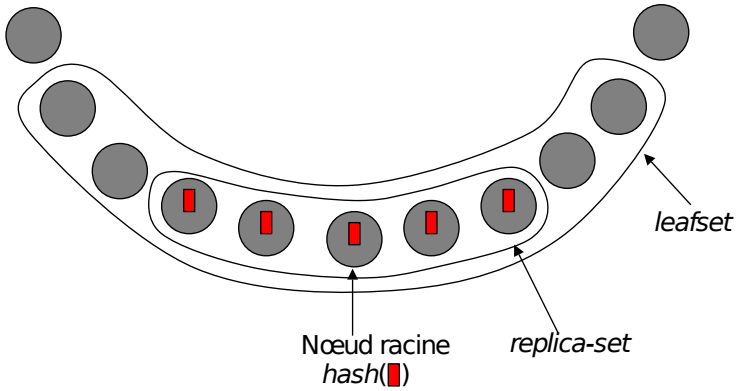
DHTs basic principles (PAST, DHASH)

- One identifier per node
  - A logical ring
  - A key-based routing (KBR)
- One identifier per data block
- Notion of *root* node
  - The one having the closest identifier
- The closest neighbors: the *leafset*
- *Good load distribution/no index*

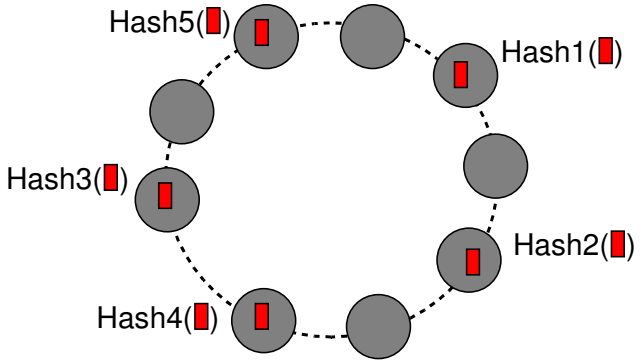


Data placement in DHTs

- Two Main families
  - Contiguous (adjacent to the root)
    Within a *leafset (leafset-based)*
  - Non-contiguous (multiple-key based, …)



- *Leafset-based* => maintenance does not depend on the number of nodes/pieces of data



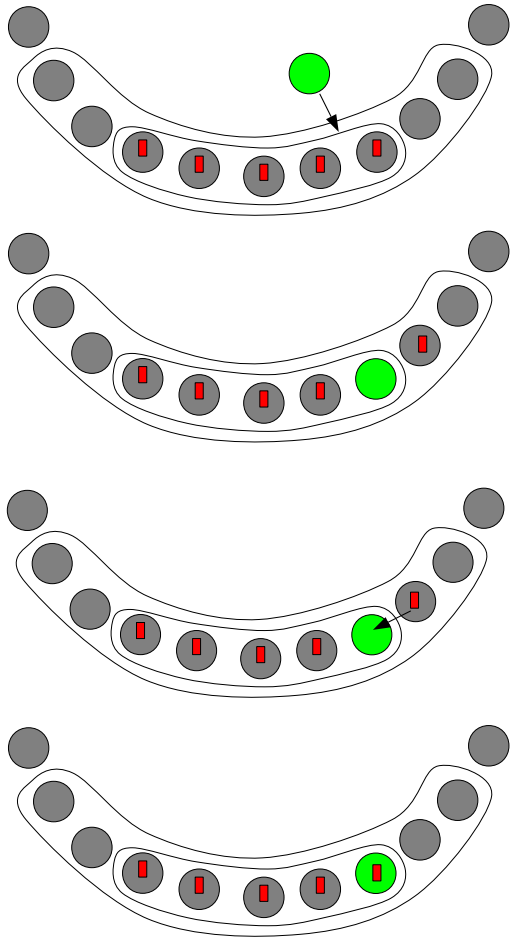
DHTs and *churn*
"useless" data movements

- Node insertion
  - New root for many data blocs
  - Within multiples replica-sets

    => Breaks the contiguousness

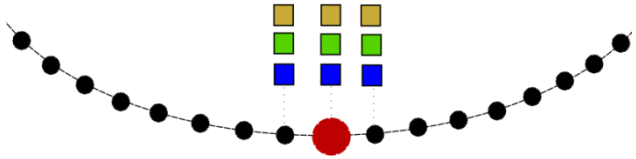
- Respect placement invariants

  => Need for data movements

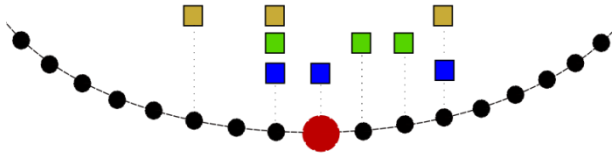


DHTs and *churn*
similar contents

- Contiguous placement
  - Similar contents on neighbor nodes
  - Few available sources for healing mechanisms

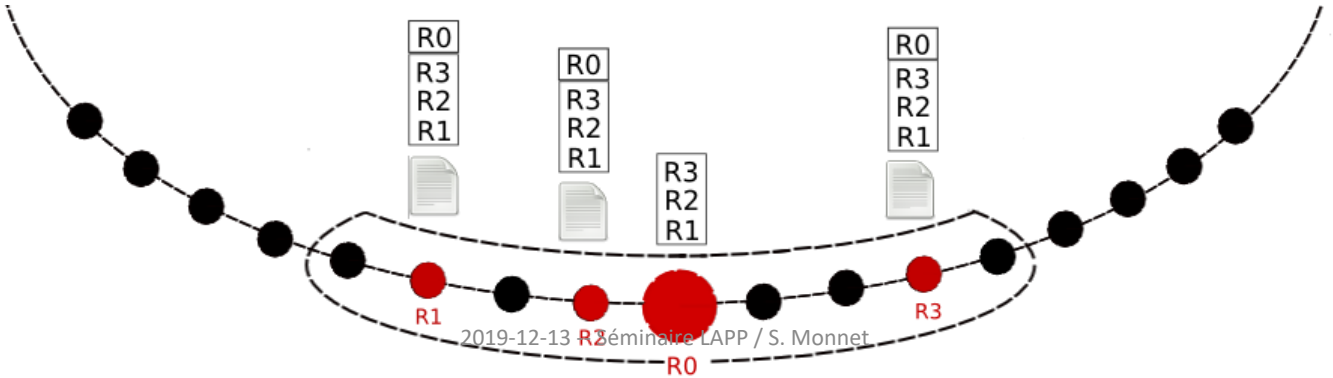


- Scattered data copies
  - Many sources and destinations when healing is needed
  - Faster reparations
  - Fewer data losses



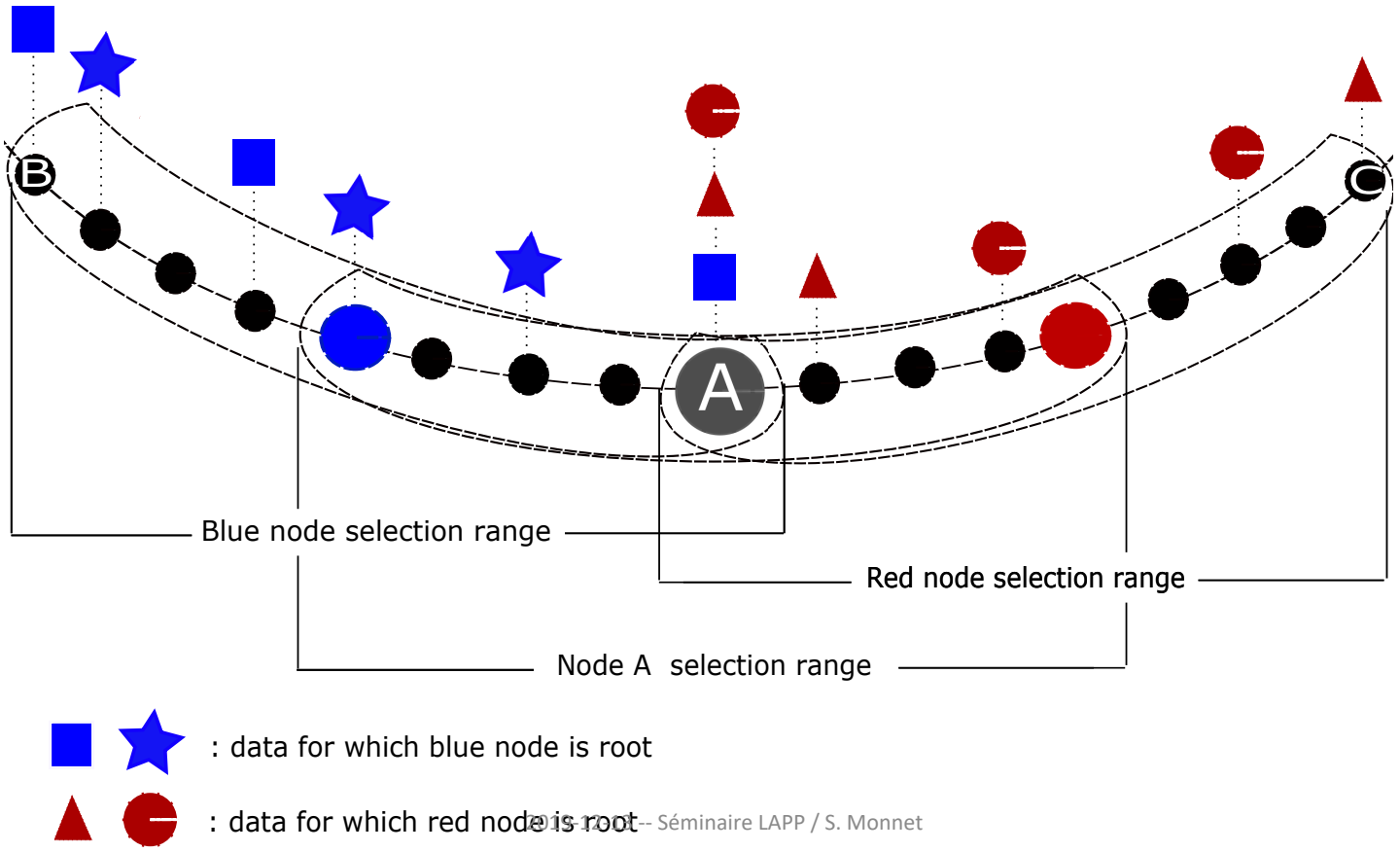
RelaxDHT : relax placement constraints

- Metadata
  - Stored by the root node and nodes hosting a copy (storers)
  - For each piece of data: list of the identifiers of the storers and the root node
- Periodical maintenance protocol
  - Root node responsible for checking storers availability
  - Storers responsible for checking the liveness of the root
  - Leafset maintenance used as a failure detector
- Gains
  - Fewer “useless” data movements
  - Best transfer parallelization



SPLAD :
Scattering and placing data

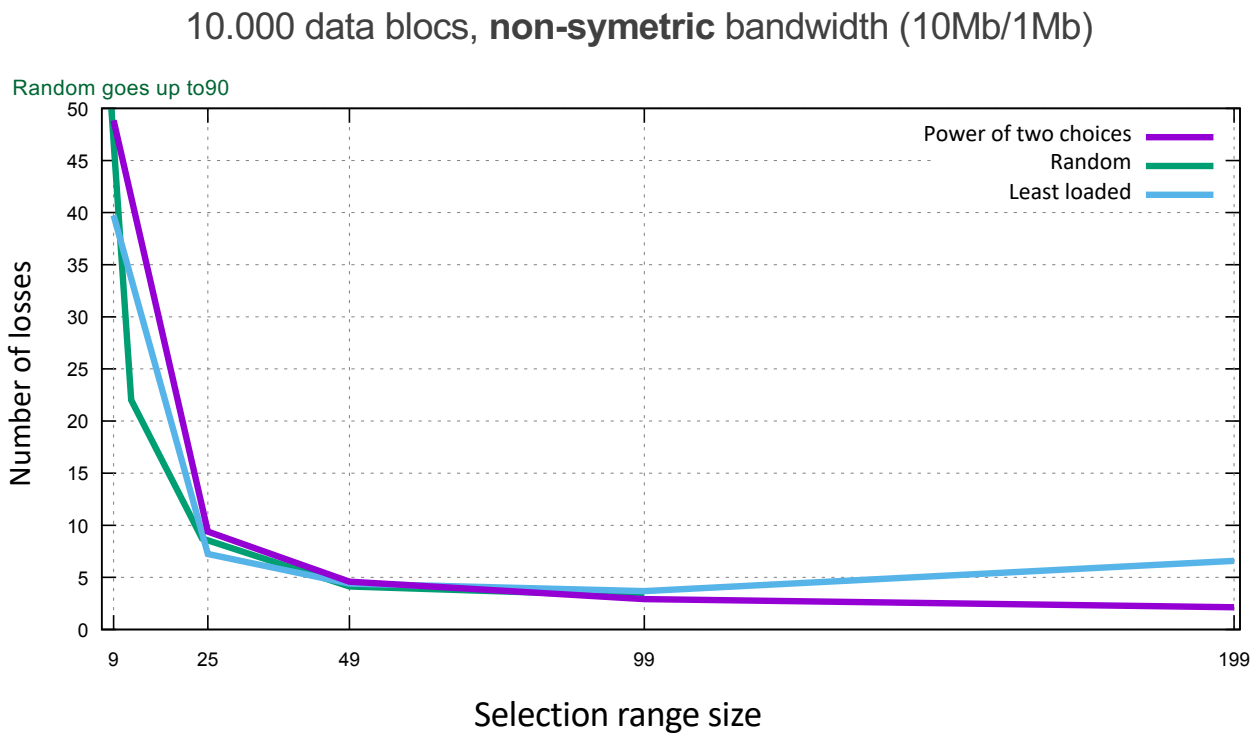
- Long term durability
- Notion of tunable *selection range*



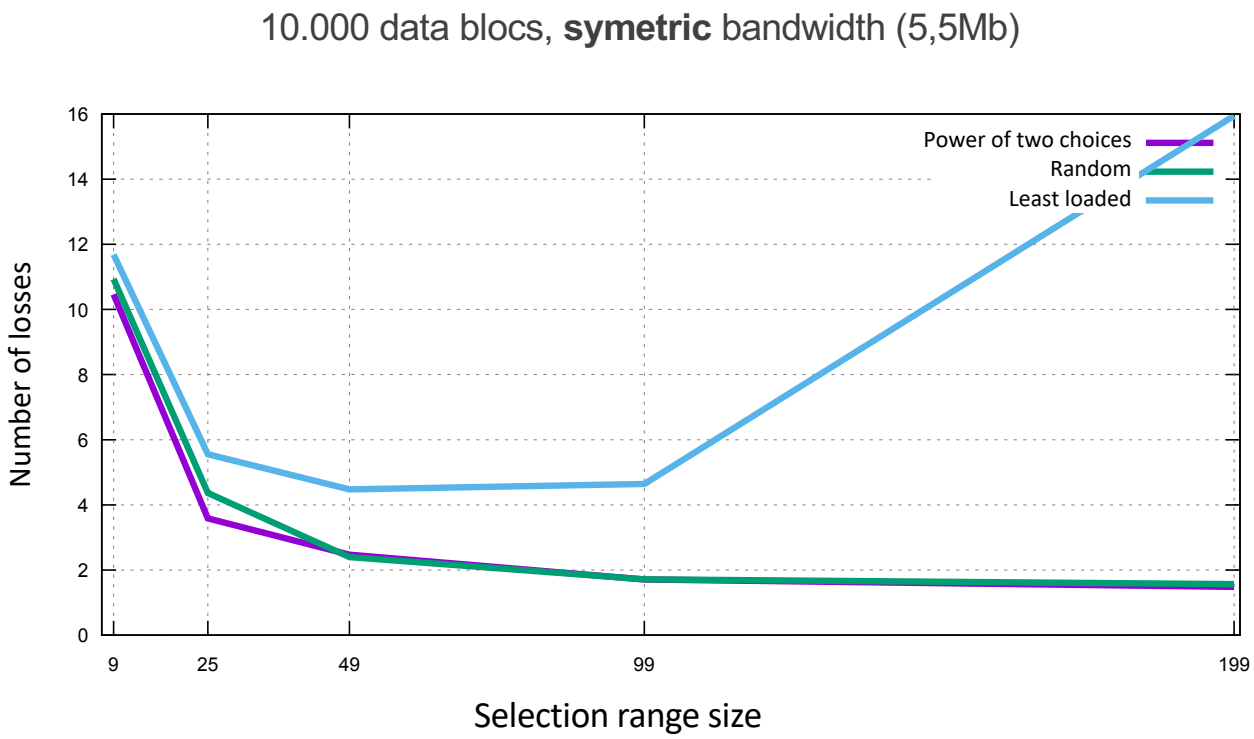
Placement within *selection ranges*

- How to choose the right storer ?
  - Random
    - Easy to implement
    - “old” nodes overloaded
  - Least loaded
    - Good storage-load distribution
    - Side effect: network congestion
  - *Power of two choices* (the least loaded between two randomly chosen)
    - Easy to implement
    - Good storage-load distribution

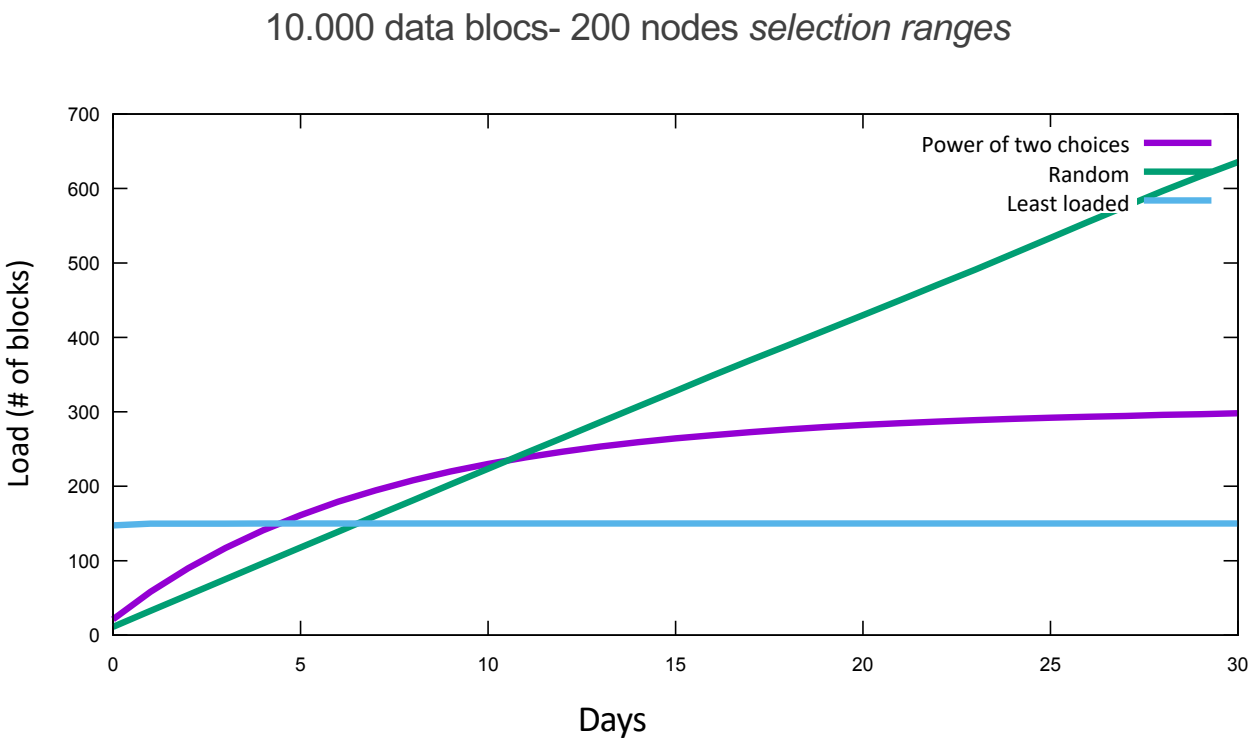
Selection range size and placement policy impact



Selection range size and placement policy impact



Storage load evolution



Impact of data placement on data durability lesons

- Relaxing data placement
  - Less useless data movements
  - Take over criteria into account (performance)
- Scattering data blocs
  - Faster healing process => fewer losses
  - Warning: maintenance cost
- Storage load distribution
  - Avoid a huge imbalance in the system
- Other factors to take into consiteration
  - Correlated failures
  - Various node reliability
  - Performance and consistency constraints

Axe 2: Access performance

- Data copies placement
  - Access locality
  - Data affinity (semantic proximity)
- Number of copies adapted to data popularity
  - Less resource waste, better load distribution among servers
- Problems
  - How to dynamically adapt according to the workload ?
  - How to efficiently locate data copies ?
  - How to deal with resource fragmentation induced by virtualization ?

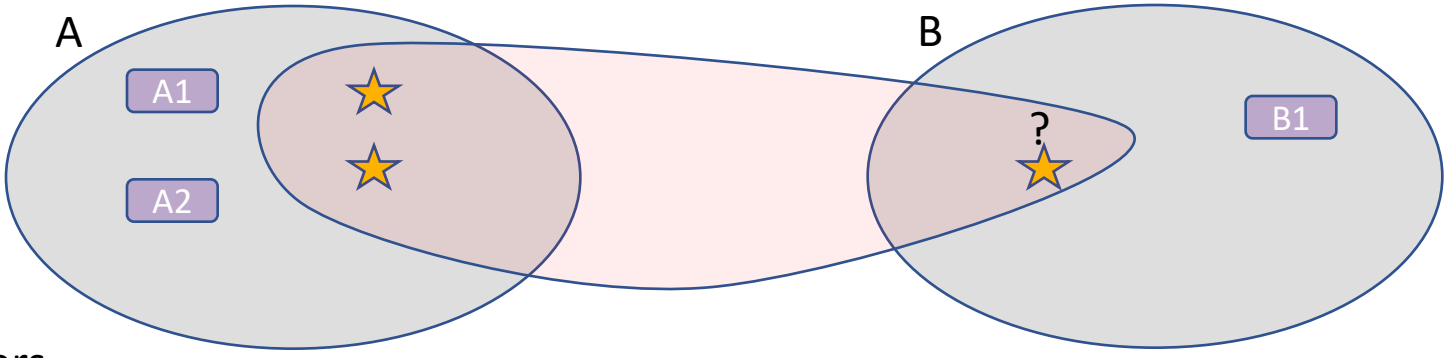
Axe 3: Replication and consistency

- Concurrent updates
  - Copies of a same piece of data can diverge
- Problems
  - How to ensure consistency of replicated pieces of data
  - Which consistency model/protocol
  - How this impacts fault tolerance mechanisms ?
  - How this impacts performance ?

CAnDoR: Consistency Aware Dynamic data Replication

Context and goal -- Etienne Mauffret thesis / ANR RainbowFS

- Context: tradeoff performance vs consistency
  - Good performance => adapted data consistency
  - Just-right consistency : synchronize only if necessary
  - Data management systems already offer multiple consistency “levels” (e.g. Casandra or Azure Cosmos DB)
- Goal: data replication should take into account
  - Access location, data popularity (cache mechanisms, CND, …)
  - Access type and frequency (access patterns)
  - Consistency protocol (strong or relaxed consistency, kind of synchronization, ...)



CAnDoR: Consistency Aware Dynamic data Replication
Approach

- Set weights and priorities
  - Are synchronizations more critical than user accesses ?
  - Are some users more important than others ?
- Maintain metrics
  - Access statistics (read / write frequencies for each user location)
  - Locally (on each storing a copy)
- Periodically compute a new replica-set
  - On a per-data basis
  - Analytically compute a new replica-set taking into account
    - The static weights - the consistency protocol
    - The “recent” (dynamic) access statistics – the access patterns

CAnDoR: Consistency Aware Dynamic data Replication
Ongoing work

- Reduce the size of the problem
  - Heuristics to reduce the number of potential candidates
- Reactivity / stability
  - Size of the time-window for the access statictics
- What are the good weights for popular consistency protocols
- Weight of the past…

Conclusions

- Data replication: a key mechanism
  - Well studied
  - Many open issues
- Need to jointly take into account
  - Fault tolerance
  - Performance
  - Consistency

Perspectives

- CAnDoR and fault-tolerance
- System level data management – explore the network/system border => Kavé Salamatian
  - Toward Kernel DHTs / NDN
  - Efficient distributed data caches
- Gadget : Toward energy-aware data management systems

ANR 2020 submission (LISTIC, LIP, LIP6, IRISA, CELESTE)