Distributed Data Storage in Computational Grids

Raphael Y. de Camargo
rcamargo@ime.usp.br

Department of Computer Science
Institute of Mathematics and Statistics
University of São Paulo, Brazil

INTEGRADE

- Middleware for Opportunistic Grids
  - Usage of idle resources, usually CPU cycles
  - Improves computing power with existing hardware
- Collaboration among 6 Brazilian universities
  - Based on free-software
  - Multi-platform
  - Uses CORBA for communication
  - Allows the execution of sequential, Bag-of-tasks, BSP and MPI applications

PROBLEM

- Applications also need storage space
  - Application input and output data, workflow applications, checkpointing data
- Where to store application data?
  - Usual solution is to store multiple replicas on dedicated storage servers
- Shared machines → unused storage space
  - Could be shared during idle periods
  - Important for institutions with limited budget

REQUIREMENTS

- Dynamic environment
  - Machines may join and leave the system
- Composed of thousands machines
  - Geographically dispersed
  - Located in different institutions
- Use only the idle period of those machines
  - Preserve the QoS of machine owner
- Need to guarantee that data will be recoverable
  - Machines may leave the system

ASSUMPTIONS

- Opportunistic grid users are willing to donate resources
  - Shared laboratories of university environments
  - Machines of researchers from institutions

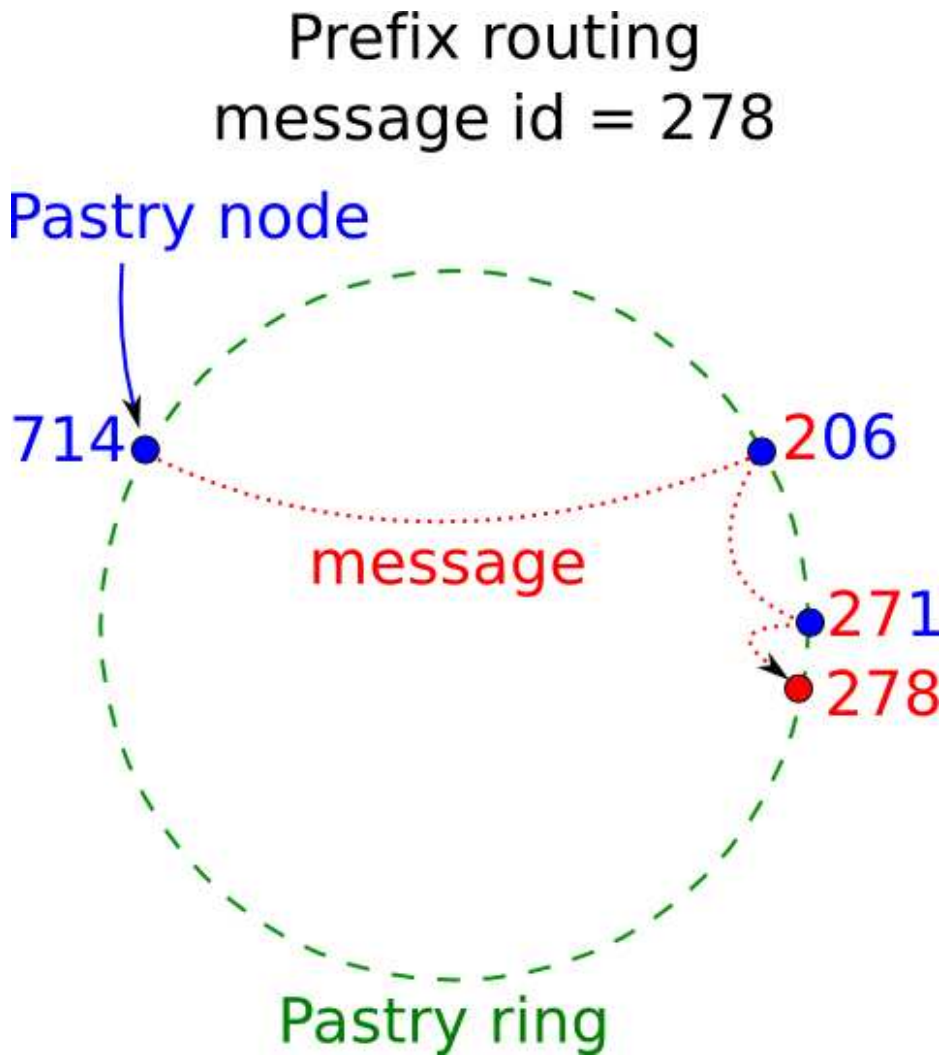
- Applications can wait a few minutes to obtain the data
  - Network delays due to bandwidth limitations
  - Machine usage patterns affect file availability

DISTRIBUTED HASH TABLES

- Distributed hash tables (DHTs) map ids to data
  - DHTs → map is distributed in several nodes
  - Each node is responsible for a range of ids
- Need to find node responsible for an id
  - Routing algorithms → target node in O(log n) hops
- Several DHTs algorithms:
  - Chord, Pastry, Tapestry, CAN
- Self-organizing
  - Allow node joining and departure

PASTRY

- Prefix routing
  - At each hop, message to node with longer common prefix
  - Node with closest Id
- Routing table
  - Allows node to find the next hop of the message
- Leafset
  - Logical neighbours



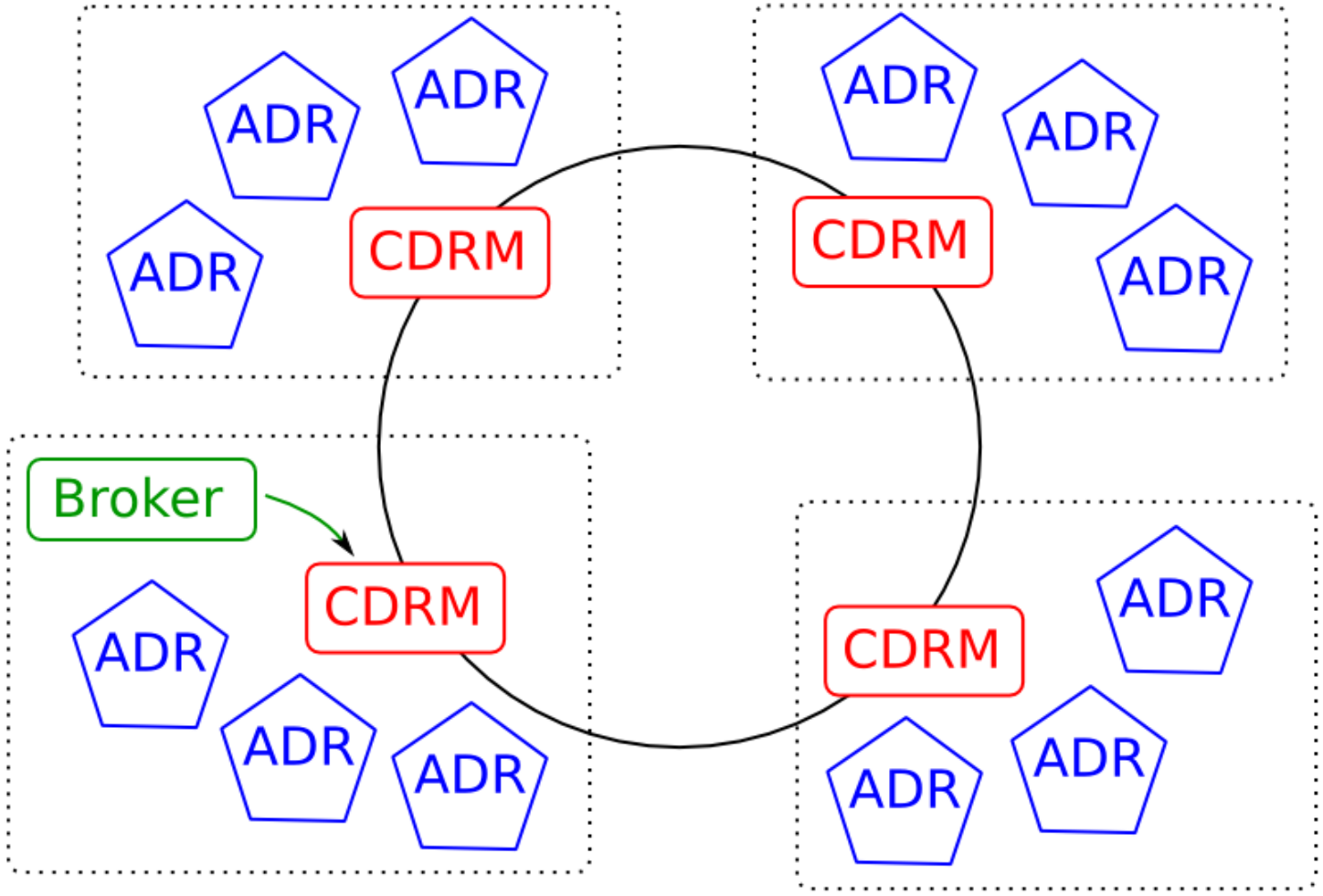
OPPSTORE

- Middleware for distributed storage of grid data
  - Provides reliable and efficient storage
  - Uses free space of shared workstation
- Organized as a federation of clusters
  - Resemble most opportunistic grids
    - OppStore clusters maps into grid clusters
  - Facilitates the management of grid dynamism

OPPSTORE (II)

- Clusters connected by a P2P routing substrate
  - Self-configuration of the cluster federation
- Files are coded into redundant fragments
  - Use erasure coding → generates *m* + *k* fragments
    - *m* are sufficient to reconstruct the file
  - Improves data availability
  - Fragments are distributed in different clusters
    - Reduces fragment unavailability caused by correlated machine usage patterns

OPPSTORE ARCHITECTURE

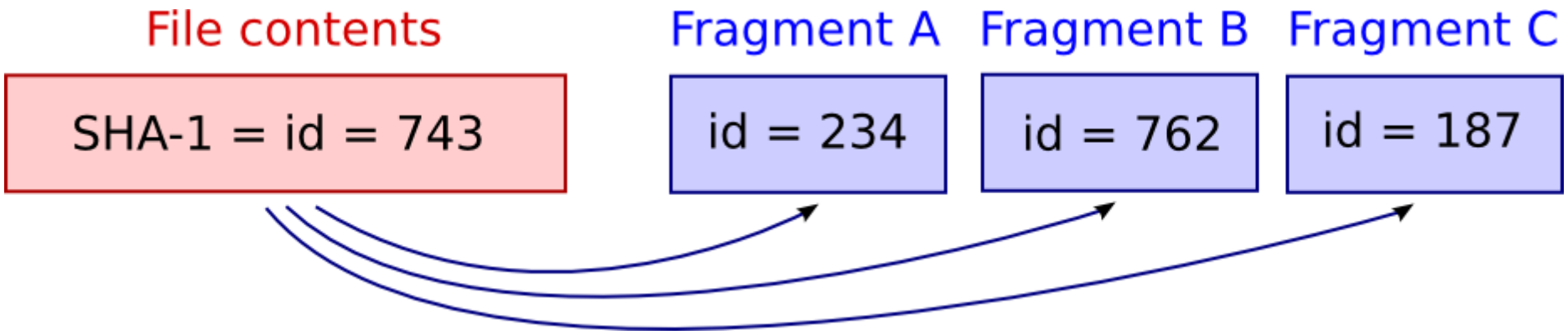


OPPSTORE COMPONENTS

- Cluster Data Repository Manager
  - Manages ADRs from an OppStore cluster
  - Connects to other CDRMs in the P2P network
- Autonomous Data Repositories
  - Executes on resource sharing machines
  - Stores grid data in the machine disk
- Access Broker
  - Allows application to access OppStore
  - Communicates with the cluster CDRM
  - Transfers data to/from ADRs

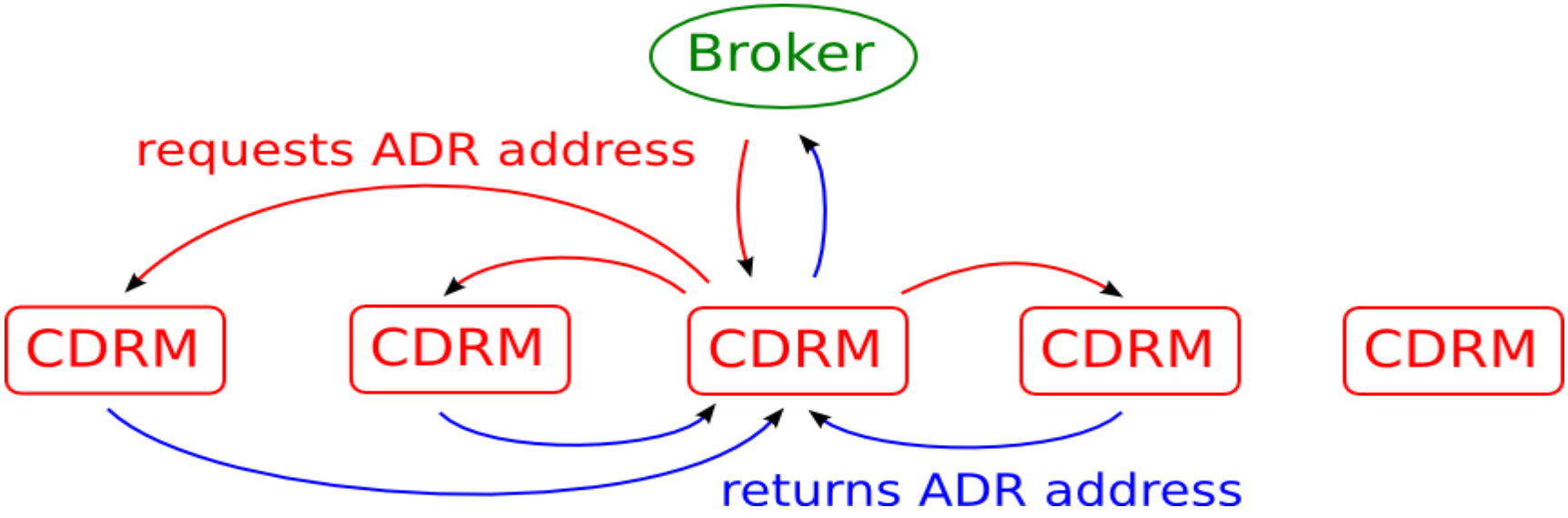
DATA STORAGE (I)

- Broker codes the file into redundant fragments
  - Example: 100MB file broken into 3 fragments of 50MB, from which 2 are sufficient to recover the file
  - Fragment secure hash → Fragment id



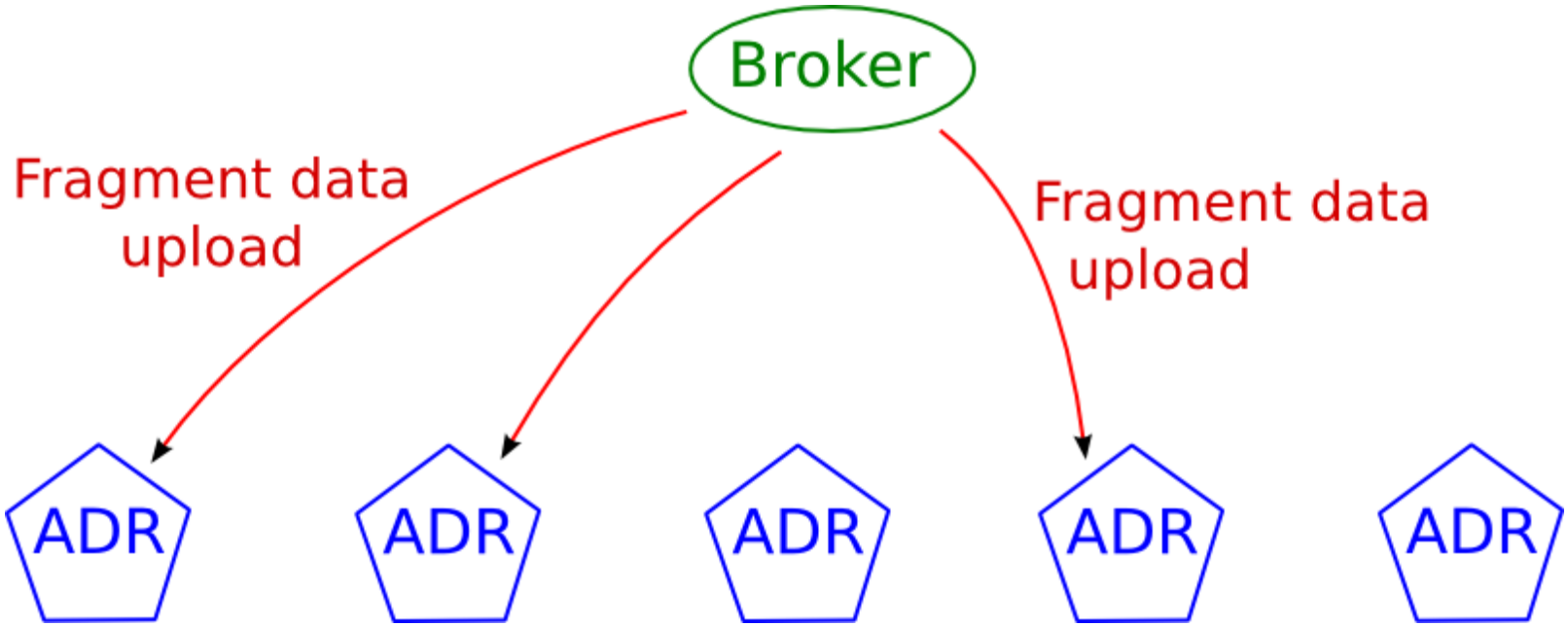
DATA STORAGE (II)

- Broker requests list of ADR addresses to CDRM
  - Sends list of fragment identifiers (hashes)
  - CDRM routes messages to target CDRMs
    - Message id ← Fragment contents hash
  - Each CDRM returns the address of an ADR



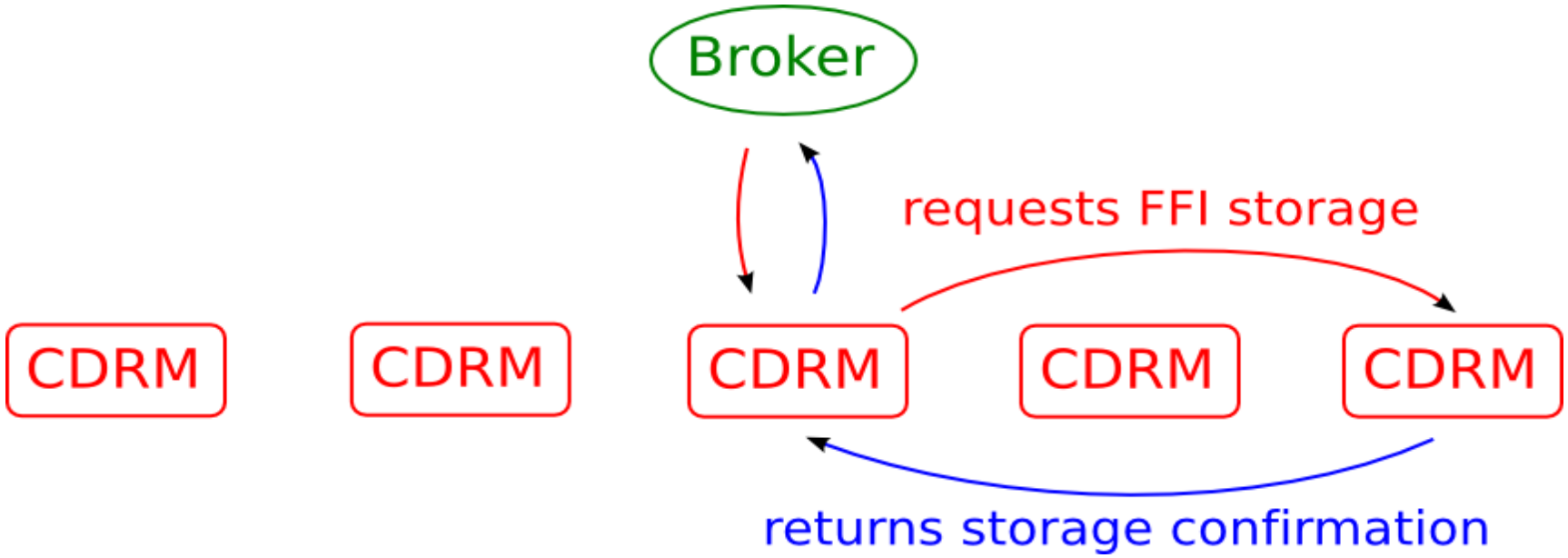
DATA STORAGE (III)

- Broker uploads fragment data to ADRs
  - Fragment data does not travel in P2P network



DATA STORAGE (IV)

- Broker constructs a File Fragment Index (FFI)
  - Contains ADR addresses and fragment hashes
  - Requests the storage of FFI to the cluster CDRM
  - CDRM routes FFI to target CDRM
    - Message id ← File contents hash



EPHEMERAL MODE

- Stores data in machines from the local cluster
  - Higher bandwidth ← Local area network
  - Lower availability ← Correlations in usage patterns
    - Not a problem when using dedicated clusters
- Applications which produce data that will be used quickly in the local cluster
  - Checkpoints, workflow applications
- Uses replication when storing files
  - Avoid coding and connection setup costs

DATA RETRIEVAL

- Broker requests FFI to the cluster CDRM
  - CDRM routes request to target CDRM
  - Target CDRM returns the FFI
- Broker download the fragments from ADRs
  - Need to download only the fragments necessary to reconstruct the file
  - Checks the integrity of fragments using their hash
- Broker reconstruct the file from fragments
  - Checks the integrity of the file using its hash

DATA MANAGEMENT

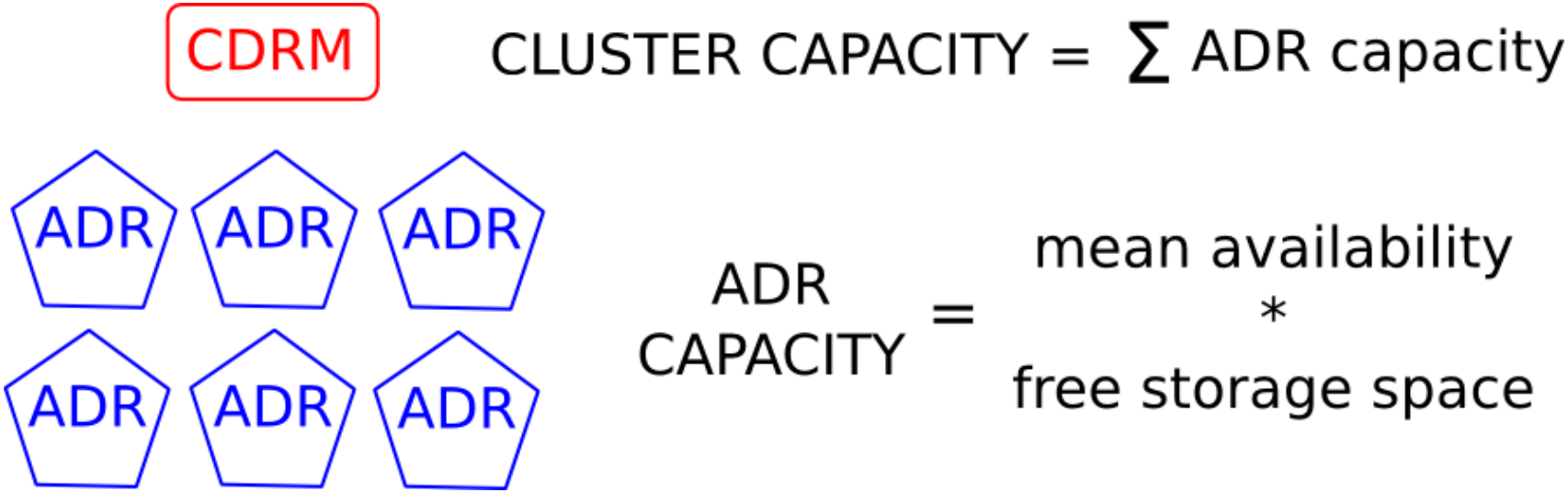
- File Management
  - Files stored for a period of time (e.g. 1 day)
  - Files stored permanently (allow later removal)
- ADRs or CDRMs can leave the system
  - Lost fragments need to be reconstructed
  - Fragment reconstruction is an expensive operation
  - Started when number of available fragments goes below a threshold

CLUSTER SELECTION

- Clusters and machines are heterogeneous
  - Use heterogeneity to improve data availability and resource usage
  - Store fragments in machines:
    - Remain idle for more time (higher idleness)
    - Have larger amounts of free disk space
- Pastry → Id range of each cluster is random
  - Independent of node characteristics
  - Better to provide larger id ranges to machines with higher idleness and free disk space

VIRTUAL IDS

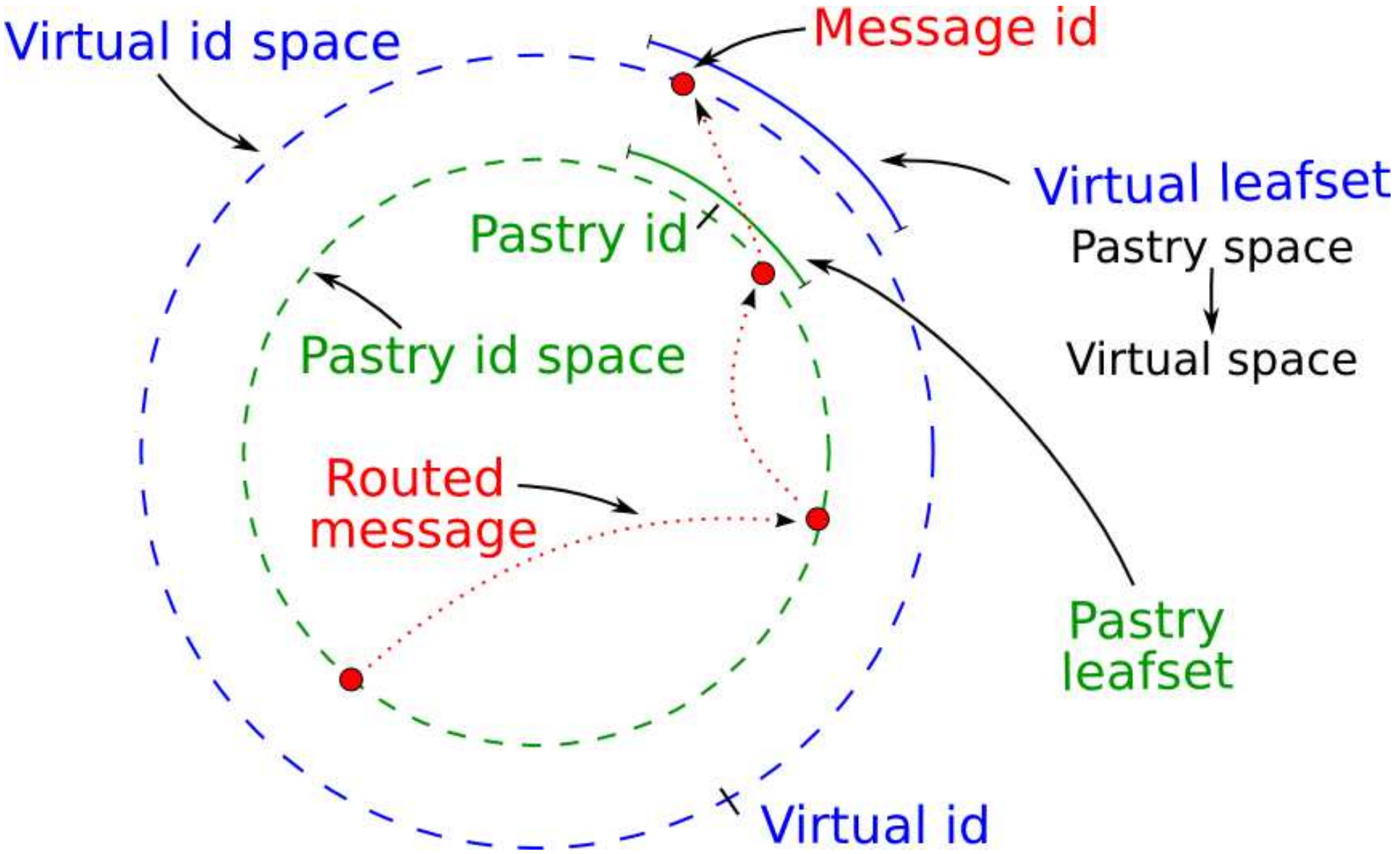
- Our extension to the Pastry protocol
- We define a new id space, called *virtual id space*
  - We assign to each cluster an extra identifier (virtual id)
  - Each cluster responsible for a virtual id range proportional to its capacity



VIRTUAL IDS PROTOCOLS

- Routing protocol
  - Allows finding the cluster with the closest virtual id
  - Use the same routing table of Pastry
  - Two extra tables of fixed size:
    - Virtual leafset and Virtual neighbourset
- Joining, departure and capacity update protocols
  - Partition of virtual id space performed locally, between a group of neighbours
- Changing the virtual id of a cluster is inexpensive
    - Need to update two tables of constant size

VIRTUAL ID ROUTING

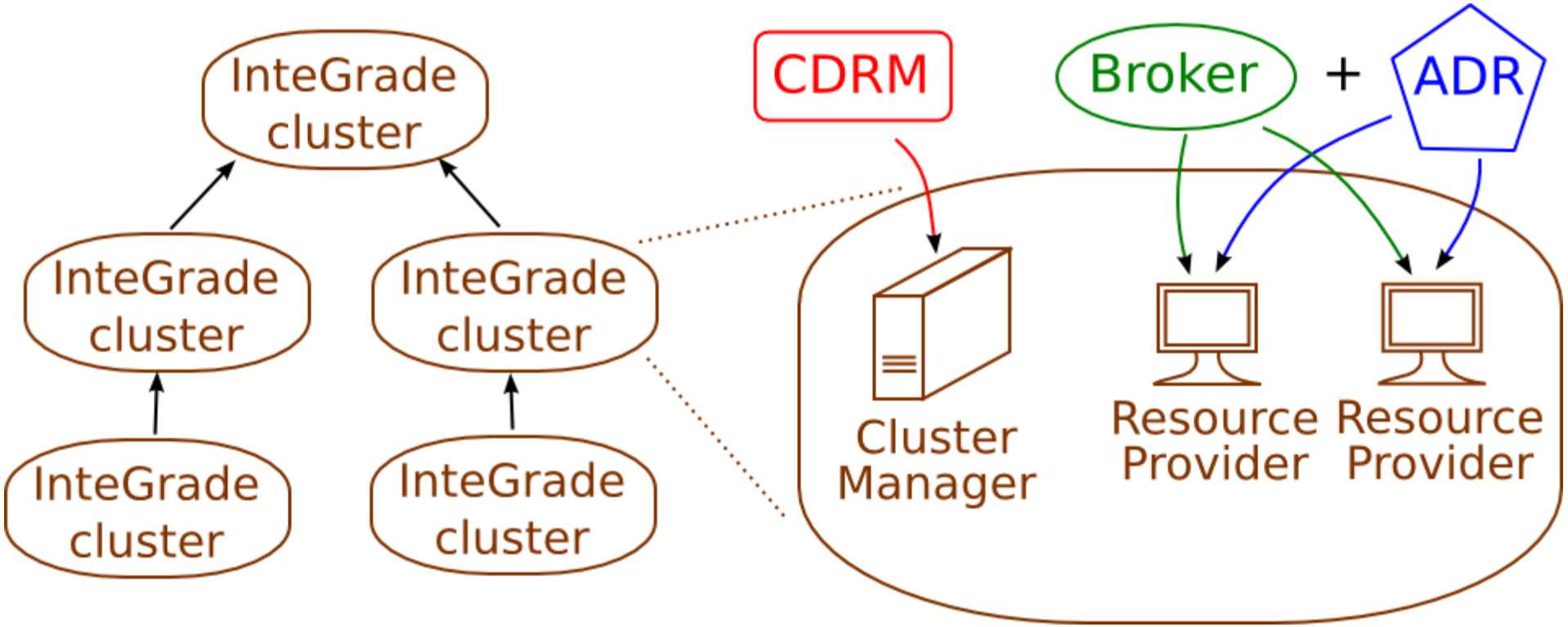


VIRTUAL IDS IN OPPSTORE

- Assign to each CDRM a virtual id
  - Virtual id range proportional to the cluster capacity
  - Cluster that will store fragments → cluster *Virtual id*
  - Cluster that will store FFIs → cluster *Pastry id*
- Virtual ids can be updated with low overhead
  - Fragments do not need to migrate
    - Fragment location is in FFI
  - FFIs do not need to migrate
    - FFIs stored and recovered using Pastry Id

OPPSTORE IN INTEGRADE

- Deployment of OppStore over InteGrade
  - Just map OppStore clusters into InteGrade clusters



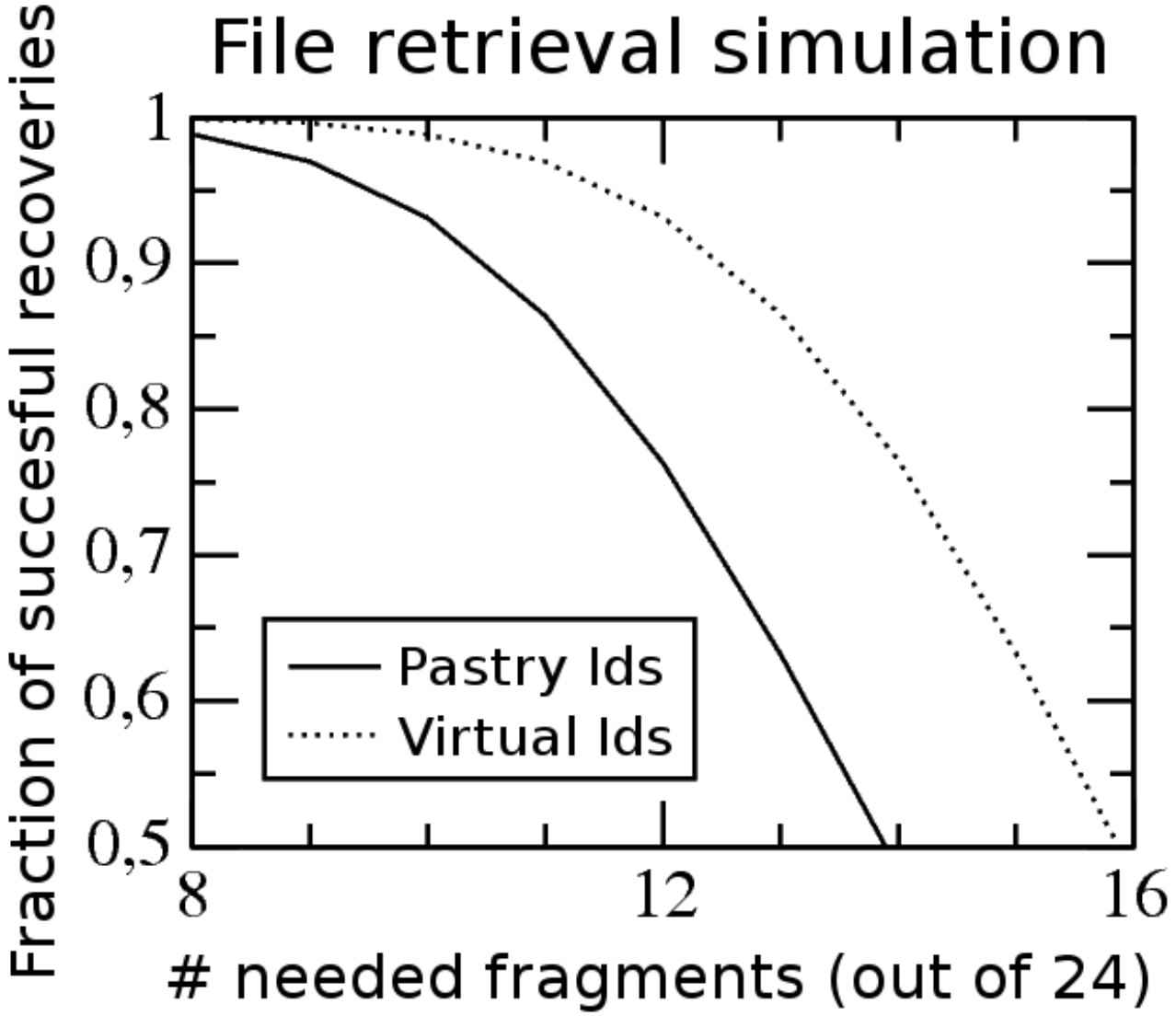
SIMULATIONS

- Simulated Grid containing 100 clusters
  - Clusters containing between 10 and 100 ADRs
  - Machine usage patterns based on real measurements*
  - Clusters distributed uniformly in 24 timezones

Daily machine usage pattern	*pattern 1*	*pattern 2*	*pattern 3*
mean day idleness	60%	80%	40%
mean night idleness	80%	40%	70%

- * BOLOSKY, DOUCEUR, ELY, AND THEIMER. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. SIGMETRICS Performance Evaluation Review 28, 1 (2000), 34–43.
- * P. DOMINGUES, P. MARQUES, AND L. SILVA. Resource usage of windows computer laboratories. In ICPP 2005 Workshops: Int. Conf. on Parallel Processing Workshops (2005), pp. 469–476.
- * M. W. MUTKA, AND M. LIVNY. The available capacity of a privately owned workstation environment. Performance Evaluation 12, 4 (1991), 269–284.

FILE RETRIEVAL

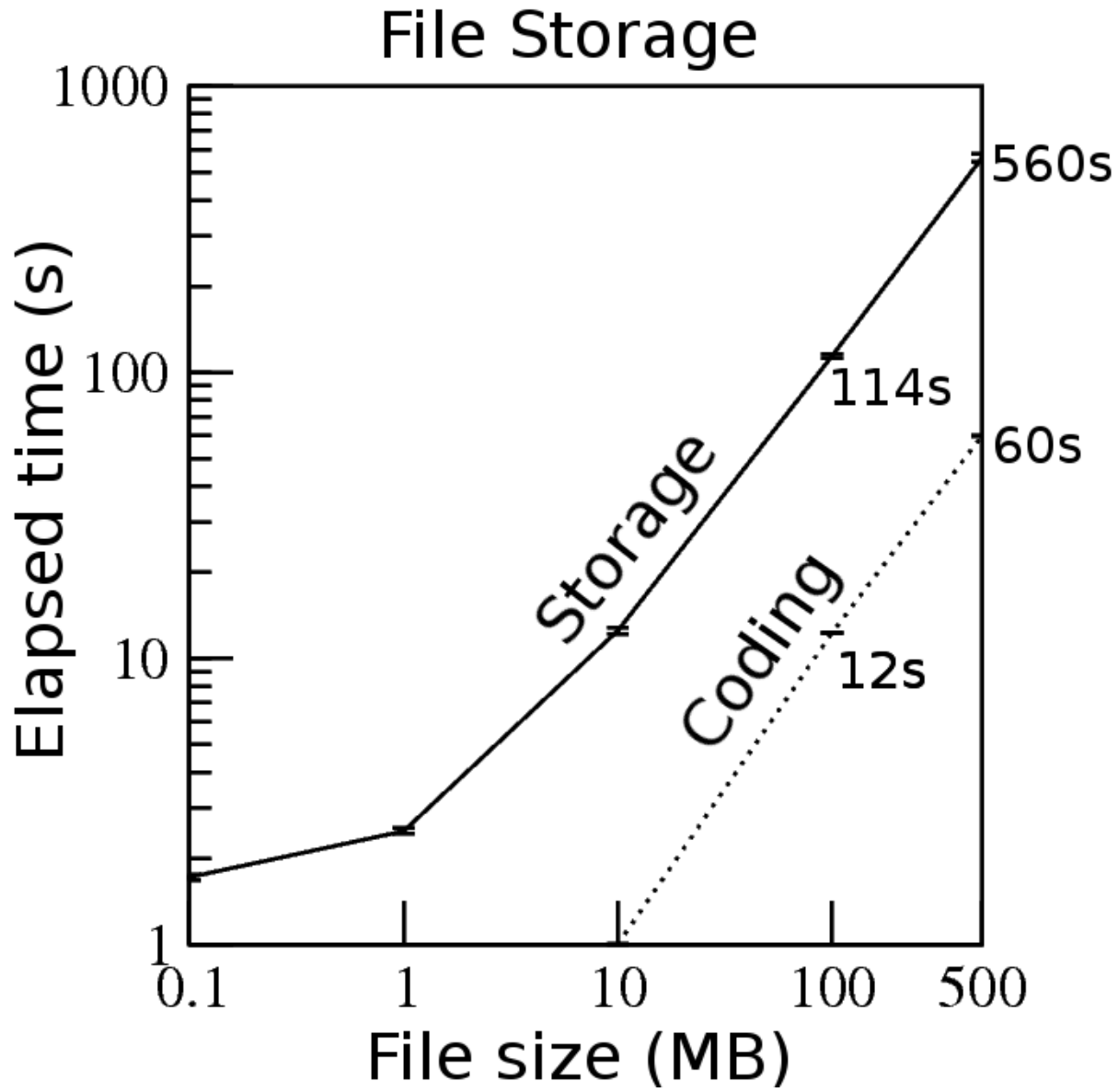


- 0.1% retrieval failures using virtual ids
- 1.2% retrieval failures using pastry Ids

EXPERIMENTAL EVALUATION

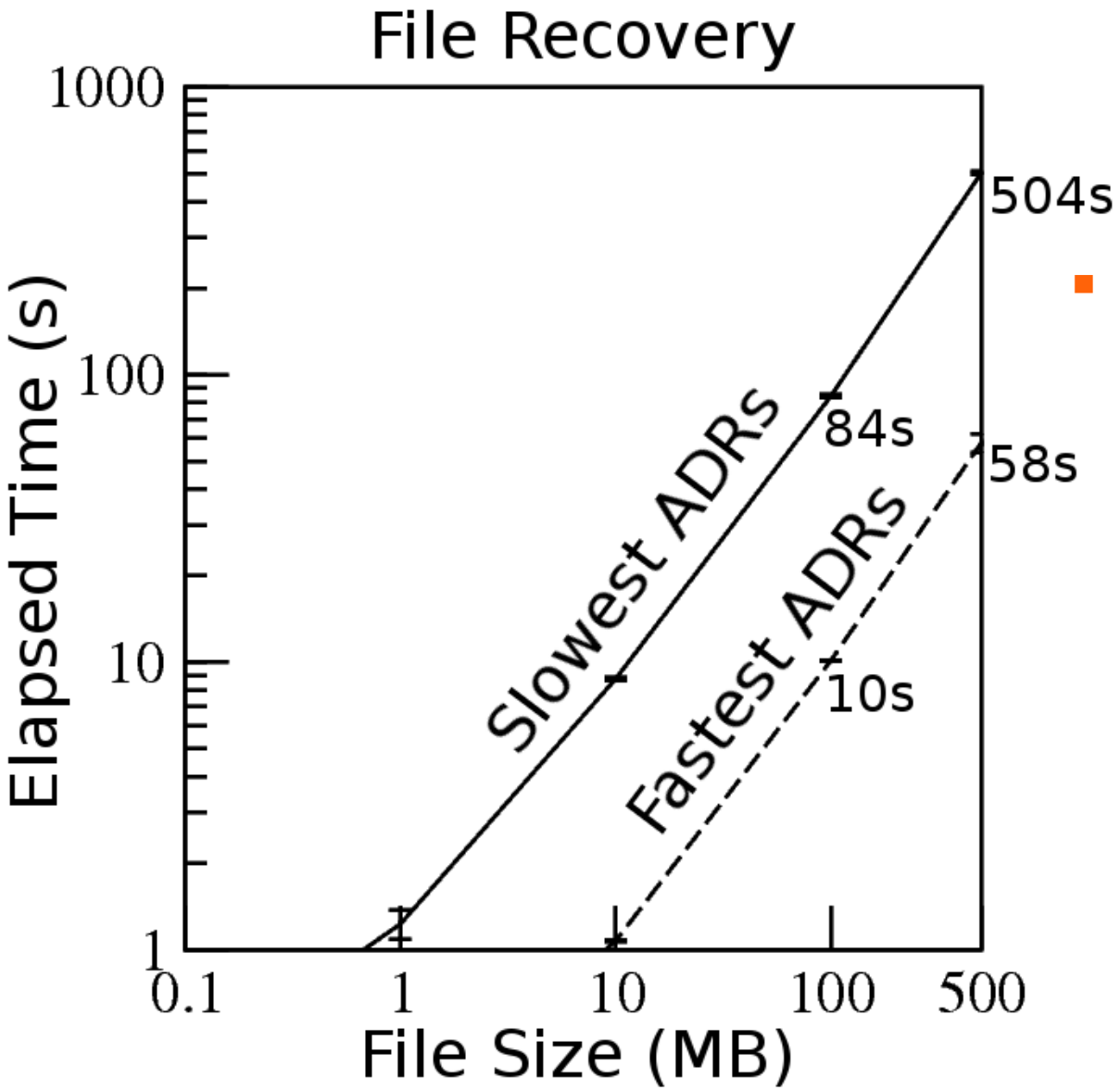
- 5 geographically dispersed clusters
  - São Paulo (3), Goiânia (1) and São Luiz (1)
  - Connected using the public Internet
- File storage and recovery experiments
  - File sizes between 100kB and 500MB
  - Coded using erasure coding:
    - 5 fragments; 2 necessary for file reconstruction
  - File storage → upload of 5 fragments
  - File recovery → download of 2 fragments

FILE STORAGE



- File storage requests:
  - Return immediately
  - Wait coding finish
  - Wait storage finish

FILE RETRIEVAL



- File retrieval requests can obtain fragments from different ADRs:
  - Fastest repositories
  - Slowest repositories

CONCLUSIONS

- OppStore → designed to provide reliable opportunistic data storage in shared machines
- Mechanisms that allows operation in dynamic and opportunistic environments
- Coding into 24 fragments (redundancy of 3)
  - Successful recovery of file in 99.9% of attempts
- Data storage and recovery speed dependent on network speed
  - Application does not need to block during storage

ONGOING WORK

- Deployment of OppStore for long periods in a real opportunistic grid
  - Check the necessities of grid users and applications
  - Improvements in the Grid interface
  - Evaluation data usage patterns
- Develop data maintenance protocols
  - The protocols should not use too much bandwidth
  - The protocols should guarantee data availability

ONGOING WORK

- Implement fault-tolerance mechanism for the infra-structure modules

- Design incentives for people to allow the usage of the computer free disk space

Raphael Y. de Camargo
rcamargo@ime.usp.br

Department of Computer Science
Universidade de São Paulo, Brazil

InteGrade project home-page:
http://www.integrade.org.br