Sistema di caching distribuito basato su DPM

G. Carlino, A. Doria, D. Spisso – INFN Napoli

A. De Salvo – INFN Roma1

E. Vilucchi – INFN Laboratori nazionali di Frascati

Workshop CCR – La Biodola – 07/06/2019

Data Management degli esperimenti LHC

- Per **ottimizzare l'uso dello storage** e **contenerne i costi**, gli esperimenti LHC, WLCG e le Funding Agencies si orientano verso il nuovo modello Data Lake basato su:
  - **Storage distribuito**
  - **Namespaces condivisi**
  - **Data Caching**
  - Ridondanza
  - Diversi livelli di *Quality of Service*
- Un numero eccessivo di storage in siti differenti porta ad inefficienze nell'amministrazione dei sistemi dovute a:
  - Necessità di manpower esperto in ogni sito
  - Maggiore onere per le "central operations" dell'esperimento
- Spesso l'analisi utente viene eseguita in cluster ospitati in siti piccoli, come i Tier3
  - È importante fornire un accesso ai dati dinamico ed efficiente anche in questi siti.

Storage consolidation e cache di dati

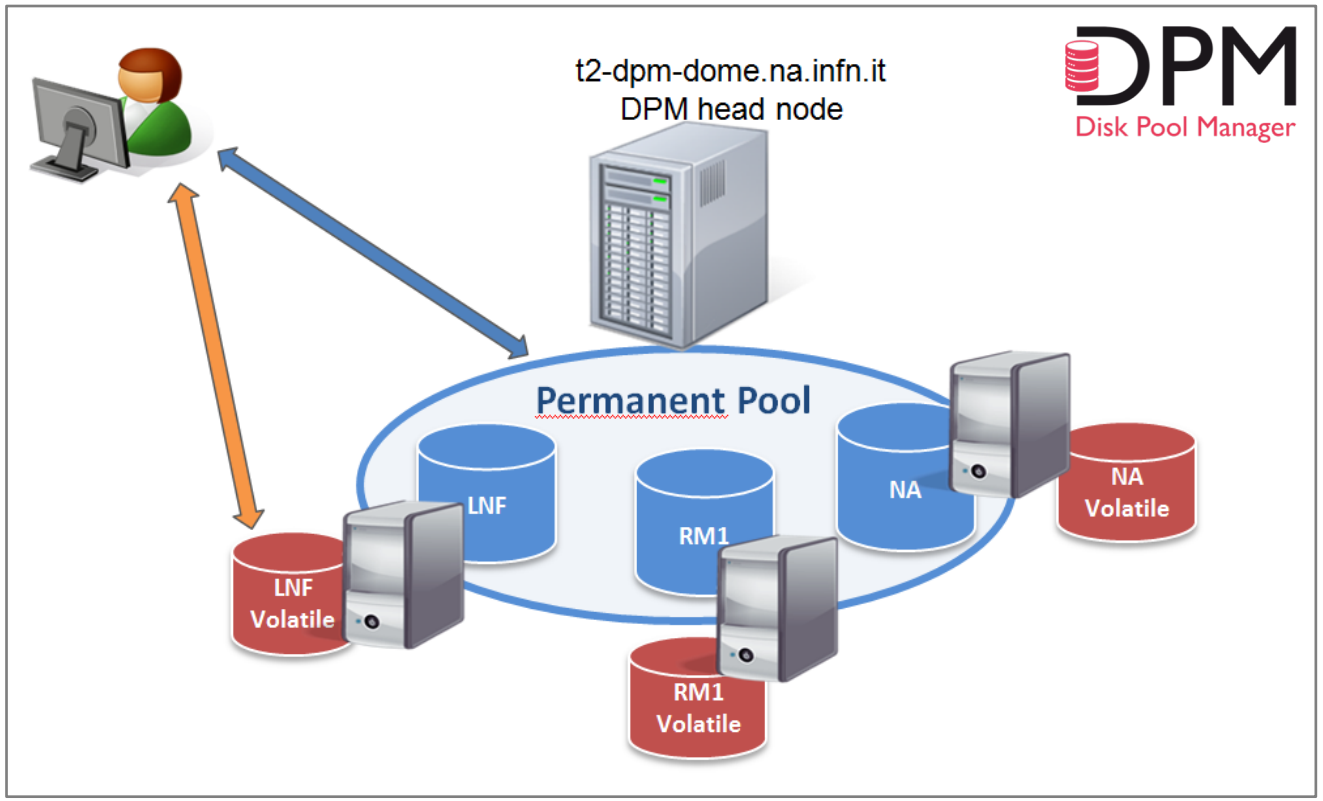
- Storage consolidation:
  - Storage federato su più siti.
  - Alleggerisce sia la gestione dello storage nei siti più piccoli, che le operations degli esperimenti.
- Caching:
  - I Data Lake prevedono delle cache
  - Ottimizzazione dell'uso delle risorse di storage, grazie ad una maggiore dinamicità: le cache si svuotano e si riempiono sulla base delle necessità degli utenti
- **Nel nostro lavoro abbiamo trattato l'aspetto dello storage consolidation e del caching**
  - Chep 2018: *Distributed caching system for multi-site DPM storage*

Prototipo di storage multi-sito con cache di dati

- Viste le necessità di ATLAS per l'accesso ai dati, nei **Tier2 di ATLAS italiani** che utilizzano **DPM** come storage system abbiamo realizzato un **prototipo di storage distribuito su 3 siti e con cache locali di dati**.
- Il sistema è stato testato in ATLAS con un reale job di analisi utente e dati di input nella cache, allo scopo di portare un'istanza di questo prototipo nell'ambiente di produzione dell'esperimento
  - Un sito primario presenta un unico namespace ed un'unica entry point per l'intero sistema distribuito
    - Incluse aree di storage (disco) dislocate in siti remoti
  - I disk pool, configurati come volatili nei diversi siti, hanno il ruolo di cache di dati locali
    - Meccanismo di accesso a zona

Configurazione: infrastruttura

- Il DPM *head node* a **Napoli** è l'unico front-end e gestisce 3 server di disco: a **Napoli**, **Roma1** e **Frascati**.
- Un pool permanente comune e 3 pool volatili:
  - Il pool permanente è distribuito ed include storage di ciascune dei 3 siti
  - I pool volatili, uno per sito, sono cache locali nei singoli siti.
- DPM con core DOME



- Un server Foreman/Puppet a Napoli per configurare e gestire DPM nei dischi remoti.
- Head e disk di Napoli e Frascati collegati a 10Gbps sulla WAN, disk Roma1 a 1Gbps.

Componenti del sistema

- Un file del **pool permanente** può risiedere fisicamente in qualunque disk node, è completamente trasparente per l'utente.
- Per richiedere un file dal **pool volatile**, invece, bisogna puntare esplicitamente al path corrispondente nel namespace, in modo da scegliere la cache locale.
  - 3 diversi path nel namespace per 3 diverse cache
- L'associazione tra il pool volatile ed il path corrispondete è fatta da **DOME** con al definizione dei **Quota Tokens**
  - Che definiscono anche la quota riservata al path associato.
- Sistemi di storage distribuiti e **tecnologie di storage eterogenee**:
  - Napoli e Frascati: standard posix file-systems
  - Roma1: storage area su *CEPH rbd* device in configurazione replica 3
- **System management e configurazione centralizzati** con un'**unica istanza Foreman/Puppet** a Napoli.

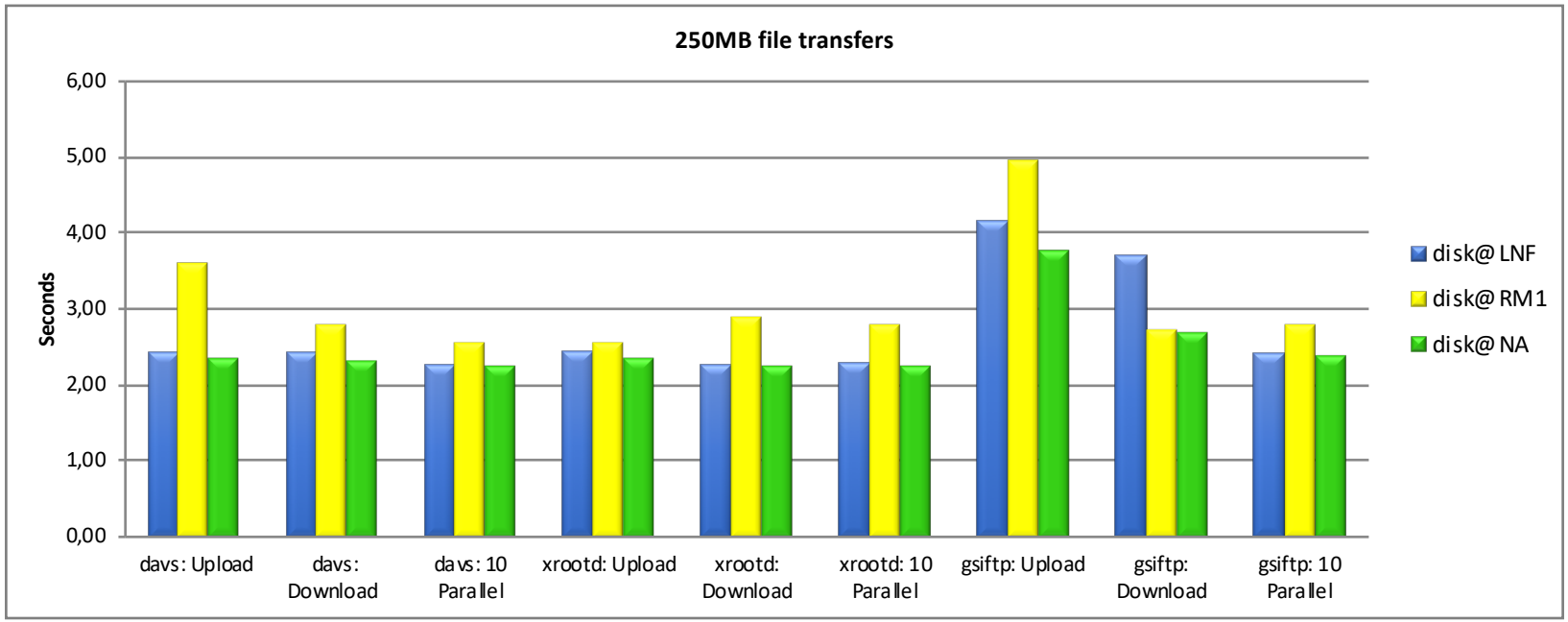
Pool volatili come cache di file

- Pool volatili che, come *lazy cache*, vengono popolati con un opportuno meccanismo in base alle richieste degli utenti.
- L'utente richiede un file alla cache e gli viene restituito in ogni caso: se il file non è presente la cache lo recupera con uno script opportuno e lo conserva per le richieste future.
- Lo script può recuperare i file per la cache da altri sistemi di storage o da federazioni come **Dynafed**.
- Nel nostro setup lo script interagisce con **Rucio Data Management** per poter accedere a qualunque storage system di ATLAS.
  - La cache agisce come un client Rucio per scaricare i file richiesti, quando è vuota viene gestita da due script innescati dalle richieste al pool volatile (*stat* per recuperare i metadati innesca uno script sull'head e *get* per accedere al file ne innesca uno sul disk node).
  - Se il file richiesto è già nel pool volatile allora non viene richiamato alcuno script e l'utente può accedere direttamente al file come avviene nel pool permanente.
- Il DM di ATLAS ignora l'esistenza delle cache che possono essere rimosse o aggiunte in qualunque momento.

Test effettuati

- Il sistema è stato testato in diverse circostanze, concentrandosi principalmente sulla **verifica delle funzionalità** piuttosto che sulla misurazione delle prestazioni poiché le prestazioni reali possono variare notevolmente in base alle connessioni di rete, all'hardware, al carico del sistema, …
  - **Test del setup distribuito**
    - Verifica che disk server su domini di rete differenti dall'head node non influisce sulla funzionalità del sistema
  - **Test della cache**
    - Test di uso dei pool volatili come cache
  - **Applicazione a casi di analisi reale**
    - Uso dei dati in cache come input per analisi di fisica, per mostrare la validità dell'approccio in termini di fattibilità e tempi di esecuzione.
- Non si sono notate differenze significative per il fatto che lo storage era distribuito su siti differenti, grazie alle connessioni di rete veloci ed affidabili
- Tutto questo è trasparente per le operazioni centralizzate dell'esperimento.

Test del setup distribuito

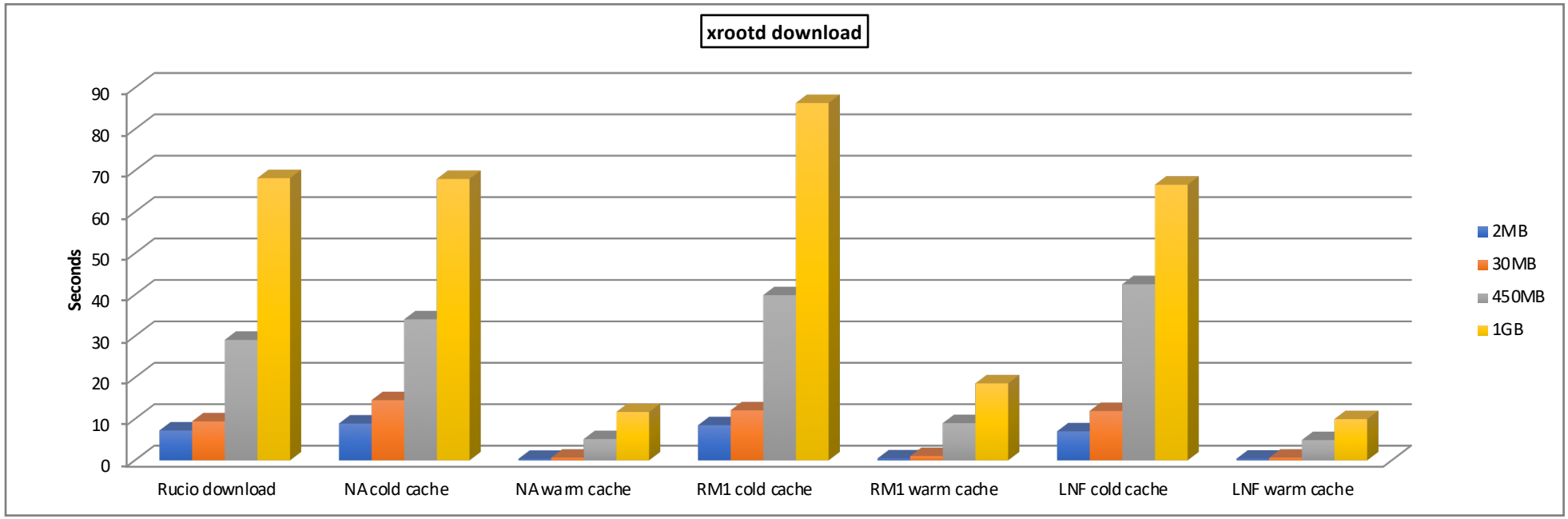


- Tempi di scrittura (upload), lettura (download) e scrittura in parallel di 10 file, per file da 250 MB.
  - Test con file di dimensioni diverse danno risultati compatibili.
- Test con protocolli diversi: **davs**, **xrootd**, **gsiftp**.
  - Eccetto gsiftp, l'effetto introdotto dal setup distribuito è trascurabile.
- Tecnologie di storage eterogenee: CEPH@RM1, posix@ NA e LNF.
- **UI** nello stesso dominio dei disk server per eliminare l'effetto del trasferimento su WAN.

Test della cache

- **L'efficacia della cache dipende fortemente da come viene utilizzata**:
  - quante volte viene riutilizzato un file,
  - quanto tempo è necessario conservare i file
  - la possibilità di riempierla in anticipo con dati popolari.
    - Nel nostro sistema non è possibile il *pre-filling*, le cache vengono popolate solo su richiesta degli utenti. Il riutilizzo dei dati dipendono fortemente dal tipo di analisi e dal workflow degli utenti locali.
- **Ciascuna area di cache è accessibile con un path specifico che deve essere noto all'utente**.
- Al primo accesso al file, la *"cold cache"* contatta Rucio per ottenerne una replica e rendere il file disponibile all'utente. Ad ogni successivo accesso allo stesso file questo è immediatamente disponibile nella *"warm cache"*.
- Sono stati fatti test di accesso ai file nel caso di cache sia *"cold"* che *"warm"*, con file di diverse dimensioni.
  - Ciascuna misurazione è stata ripetuta 10 volte, poiché i risultati potrebbero variare in base al traffico di rete e al carico dei disk server.

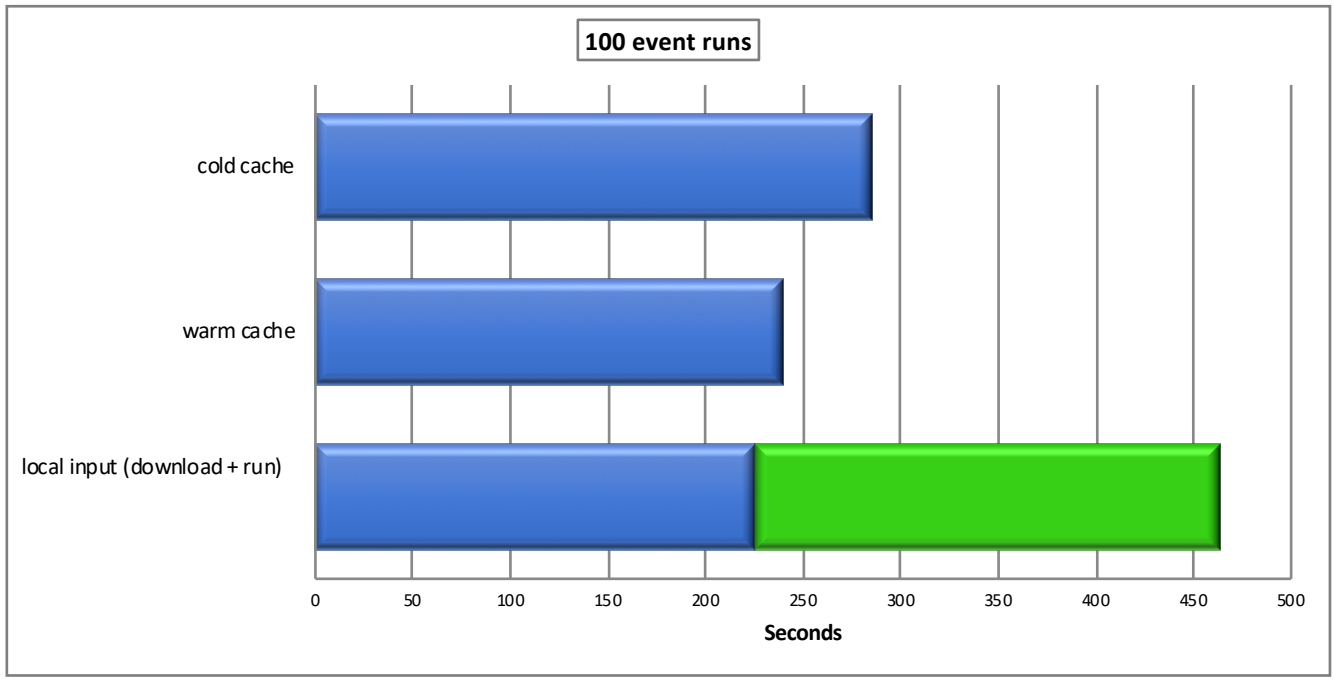
Tempi di trasferimento dalla cache



- I tempi per un “rucio download” direttamente dalla grid sono molto simili al trasferimento dalla “cold cache”, mentre il tempo di trasferimento dalla “warm cache” è molto minore.
- **Confrontando Napoli e Frascati si verifica nuovamente che l’overhead dovuto al disco remoto è trascurabile**. Un aumento dei tempi si ha a Roma1 che ha una diverso setup.
- I test sono stati ripetuti con **diversi protocolli** (davs, xrootd, gsiftp), ma i risultati non sono significativamente differenti.

Test di file caching in uno use-case reale

- Test con un job di ATLAS reale che legge i file di input dalla cache con accesso diretto xrootd.
  - File di input di 1.8 GB che contiene 55k events, il programma gira su un sottoinsieme di 100 eventi.
  - Ogni misura ripetuta 10 volte
- Tempi per esecuzione del job nei 3 diversi casi per il file di input:
  - Primo accesso alla cache, *cold cache* deve accedere al file in grid prima di fornirlo all'utente.
  - *Warm* cache, il file è già presente in cache alla richiesta dell'utente.



- L'utente scarica l'intero file di input dalla grid sul file system locale ed esegue il job sulla UI locale.

Vantaggi del file caching in uno use-case reale

- L'utente non ha bisogno di scaricare precedentemente il file di input in un'area locale per eseguire poi il lavoro a download completato.
- Riutilizzabilità dei file in cache infinite volte, da parte di qualunque utente e da qualsiasi risorsa di calcolo del sito.
- Anche in caso di *cold cache* il tempo necessario ad eseguire il job è generalmente minore dell'approccio senza uso della cache.
- Cosa manca:
  - Non esiste ancora un meccanismo di pinning per mantenere nella cache i file più acceduti, i dati vengono rimossi dalla cache quando è piena e l'utente non ha alcun controllo sulla lifetime dei file.

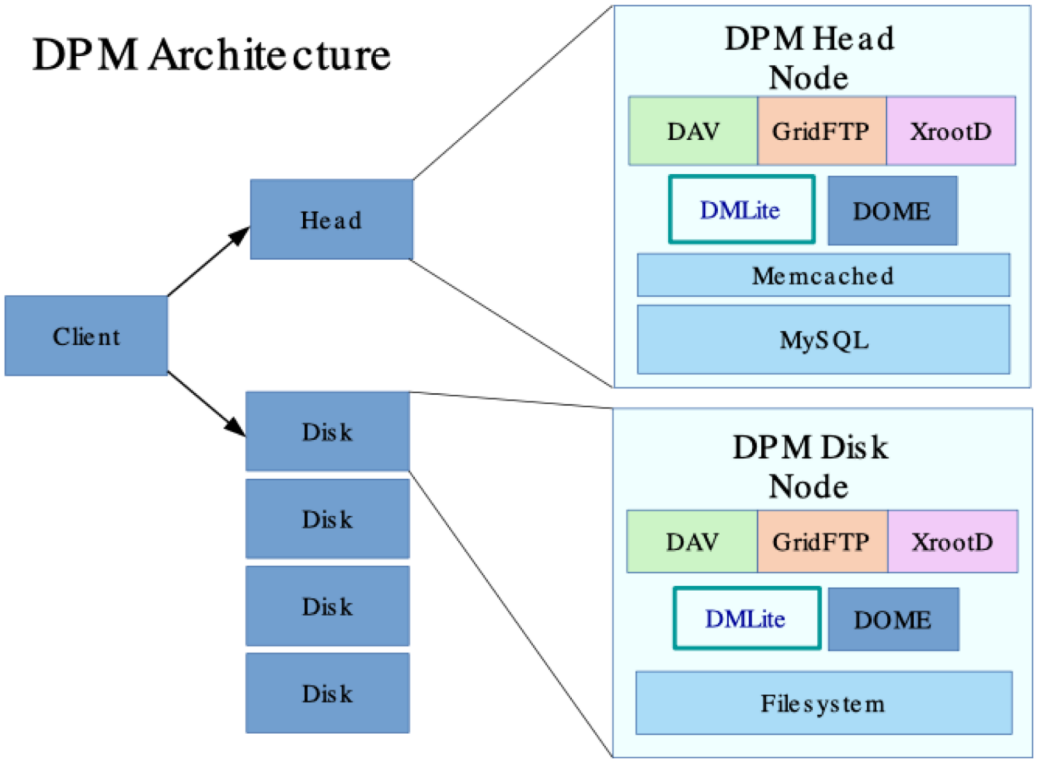
Conclusioni

- **I test hanno mostrato la fattibilità dell'architettura proposta e la sua funzionalità.**
  - I risultati dei test di trasferimento mostrano che storage con tecnologie differenti e geograficamente distribuiti non influenzano in modo significativo l'accesso e il trasferimento dei dati.
  - Un sito può ospitare solamente dell'hardware, demandando del tutto la gestione.
    - Sistema adatto a siti piccoli come **Tier3 diskless**.
  - Si ottiene in generale uno snellimento delle operations dell'esperimento.
- Il Sistema presenta alcune delle caratteristiche di un modello Data Lake:
  - **Storage distribuito**
  - **Namespaces condivisi**
  - **Data Caching**
- Si prevede di proseguire la sperimentazione nell'ambito del progetto IDDLS.
- **Prossimi passi:**
  - **Estendere il sistema ad una infrastruttura più ampia e con storage più etereogenei**
  - **Integrare il sistema nell'ambiente di ATLAS di produzione.**

Backup

DPM with DOME

- DPM is a simple way to create a disk-based grid storage element composed by many disk servers. DPM supports the data and metadata access protocols that are relevant for file management and access in a distributed environment (HTTP, WebDAV, xrootd, SRM, gsiftp).
- The system is composed of one Head Node, that hosts the metadata and manages the requests, and several Disk Nodes hosting data files and managing the file transfers with the mentioned protocols
- Traditionally head and disk nodes are located at the same site



Different DPM paths

- A file of the **permanent pool** can physically reside in any of the disk nodes
  - completely transparent to the user
  - ATLAS is aware of these files
  - *t2-dpm-dome.na.infn.it:/dpm/infn.it/home/atlas /fileA*
- To get a file from one of the **volatile pools** instead, we have to explicitly point to the corresponding path in the namespace to be able to choose the local cache.
  - The paths for the three caches (ATLAS not aware):

    t2-dpm-dome.na.infn.it:/dpm/na.infn.it/home/atlas /fileB

    t2-dpm-dome.na.infn.it:/dpm/roma1.infn.it/home/atlas/fileC

    t2-dpm-dome.na.infn.it:/dpm/lnf.infn.it/home/atlas/fileD
- The association between volatile pools and corresponding storage paths is made in **DOME** by the definition of **Quota Tokens**
  - also define the space quota reserved to the associated path.

Caches as Rucio client

- The cache simply acts as a Rucio client to download files. When the cache is not yet populated, its behaviour (implemented via DOME) is driven by two scripts that are triggered by the file requests toward a volatile pool. Such requests can be *stat* (retrieve the metadata) or *get* (download or access the file):
  - the ***stat*** call triggers only the script on the head node which checks if the file exists in the data source and retrieves the size and the metadata as well.
  - the ***get*** request triggers a pull script that runs on the disk node corresponding to the chosen volatile pool. Such script can address any system as file source and can use any implemented protocol for the file transfer.
- While the transfer of the file from the chosen source to the volatile pool is in progress, DOME returns to the client the *accepted* status code (202 for the dav/davs protocol), asking to wait until the completion of the transfer. When the file is ready in the volatile pool, DOME returns the *succeeded* status code (200 for the dav/davs) and the transfer to the requesting client is started.
- Most of the file transfer clients and libraries, used in grid environment, support the 202 status code, so the client is able to wait until the file is copied to the cache. For example, in our tests we have used gfal client, for which we can set the retry-delay and the max retry attempts in case of 202 status code.

Caches

- In case the volatile pool is already populated and a cached file is requested, no script is called. The users can download or access the files directly, in the same way as for the permanent pool.
- We have chosen to fill the cache using as data source all the ATLAS files registered in Rucio. We implemented and tested a retrieving script that runs a Rucio client: when the script is triggered, Rucio is initialized and a download operation is started to the local physical destination directory (which is mapped to the volatile pool) specified by DOME.
- The scope and the file name are supplied by the users through the LFN used for the get request and the virtual directory named after the Rucio scope is created and populated with the required file. For example, to get from the cache the file

  mc15_13TeV:DAOD_EXOT5.1053164.pool.root.1

  the following request has to be issued to the volatile pool:

gfal-copy davs://t2-dpm-dome.na.infn.it/dpm/na.infn.it/home/atlas/mc15_13TeV/DAOD_EXOT5.10531640.pool.root.1 ./