UNICORE DISTRIBUTED STORAGE MANAGEMENT SERVICE

Tomasz Rękawek, Piotr Bała, Krzysztof Benedyczak

Goals

- Distributed storage with the SMS interface
- Consisted of the ordinary SMS storages
- File transfers directly from a client to destination SMS (without any proxy)
- Usage of the standard UNICORE authorization and authentication mechanisms
- Simple architecture and easy installation

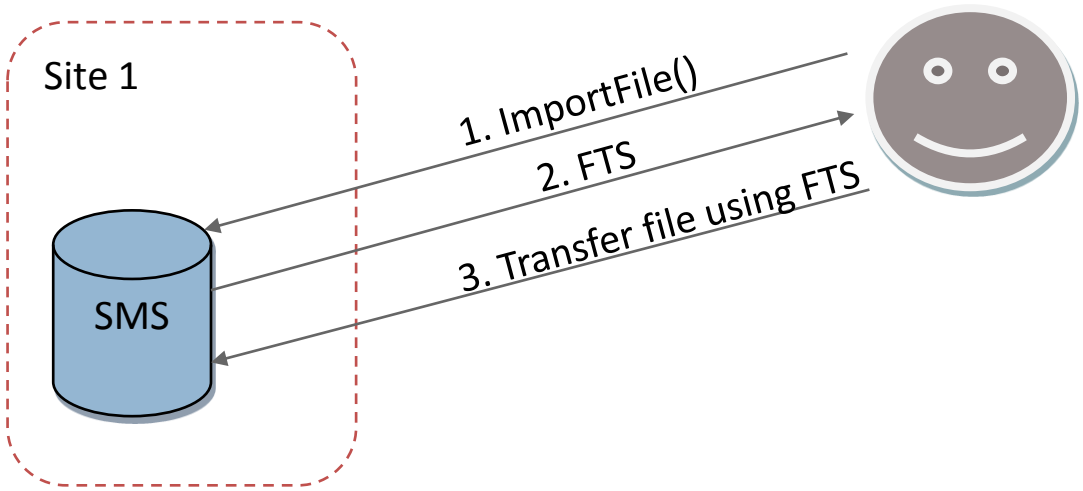
Existing solutions

- Chemomentum DMS
  - Combines many storages into one, but…
  - Has custom client interface (different from SMS)
  - Is integrated with additional and not necessary services like Ontology Service which are hard to separate
  - All transfers are done using central Data Management System Access Service
- UniRODS and UniHadoop
  - Interfaces to external distributed storage software (iRODS and Apache Hadoop)
  - That additional software has to be installed on all disk servers
  - iRODS and Apache Hadoop have their own authorization mechanisms
  - Internal transfer may be insecure

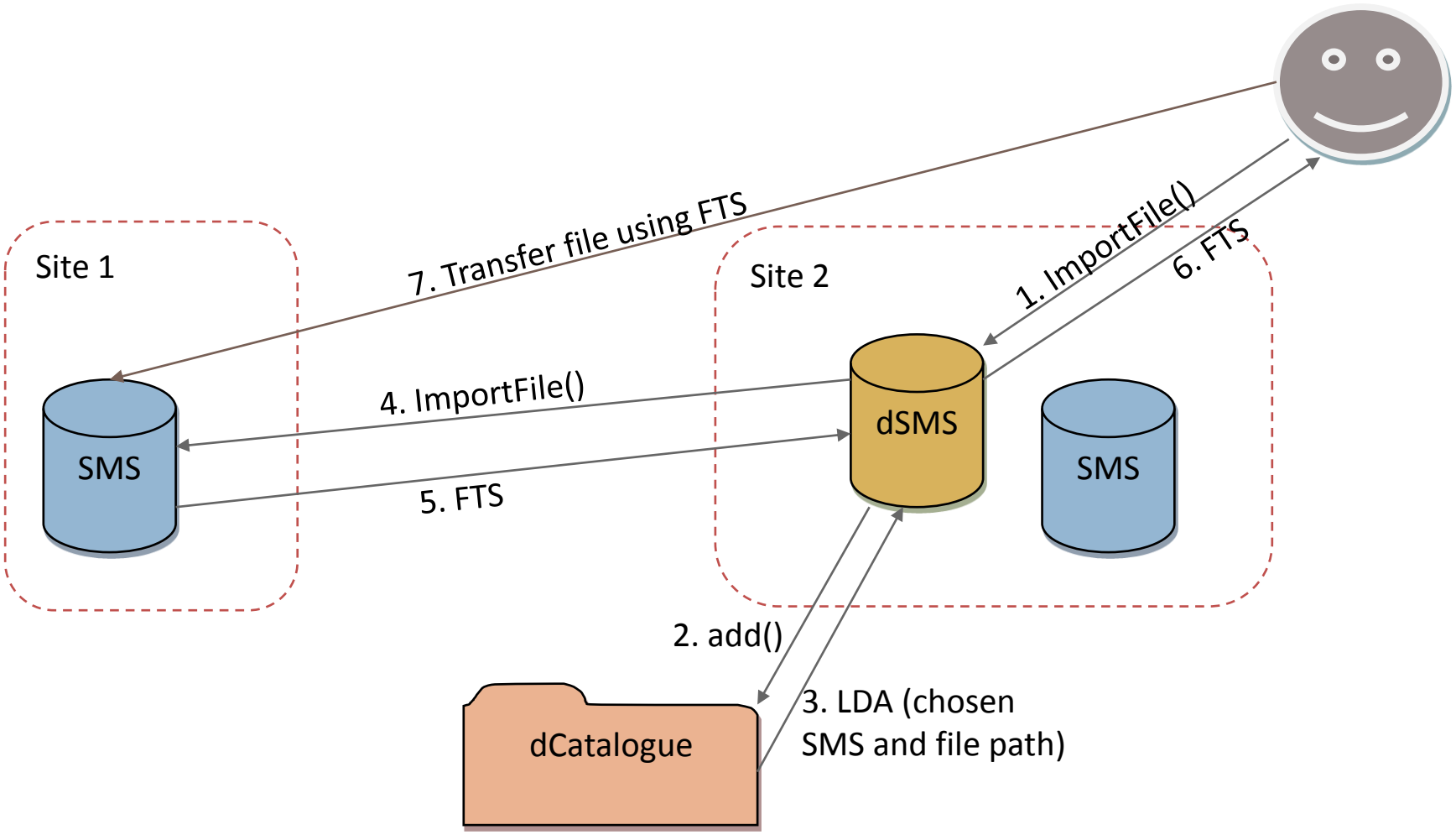
New solution: dSMS

- New component in the existing UNICORE/X containers called dSMS (entrypoint)
- New service in grid – dCatalogue contains file locations
- Files are stored on the existing SMS storages
- Clients connect to any dSMS and get access to the same shared space

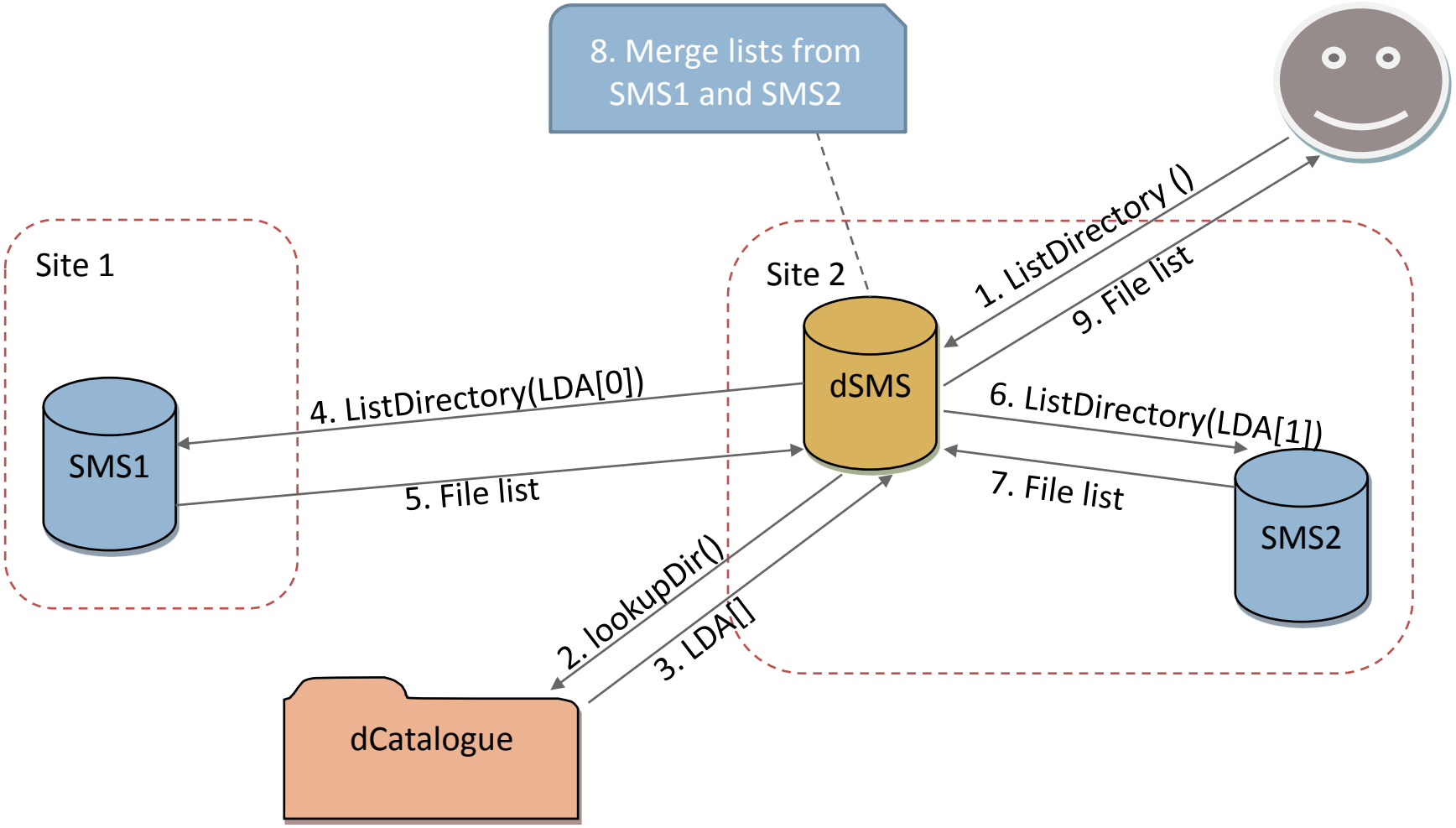
ImportFile operation in SMS



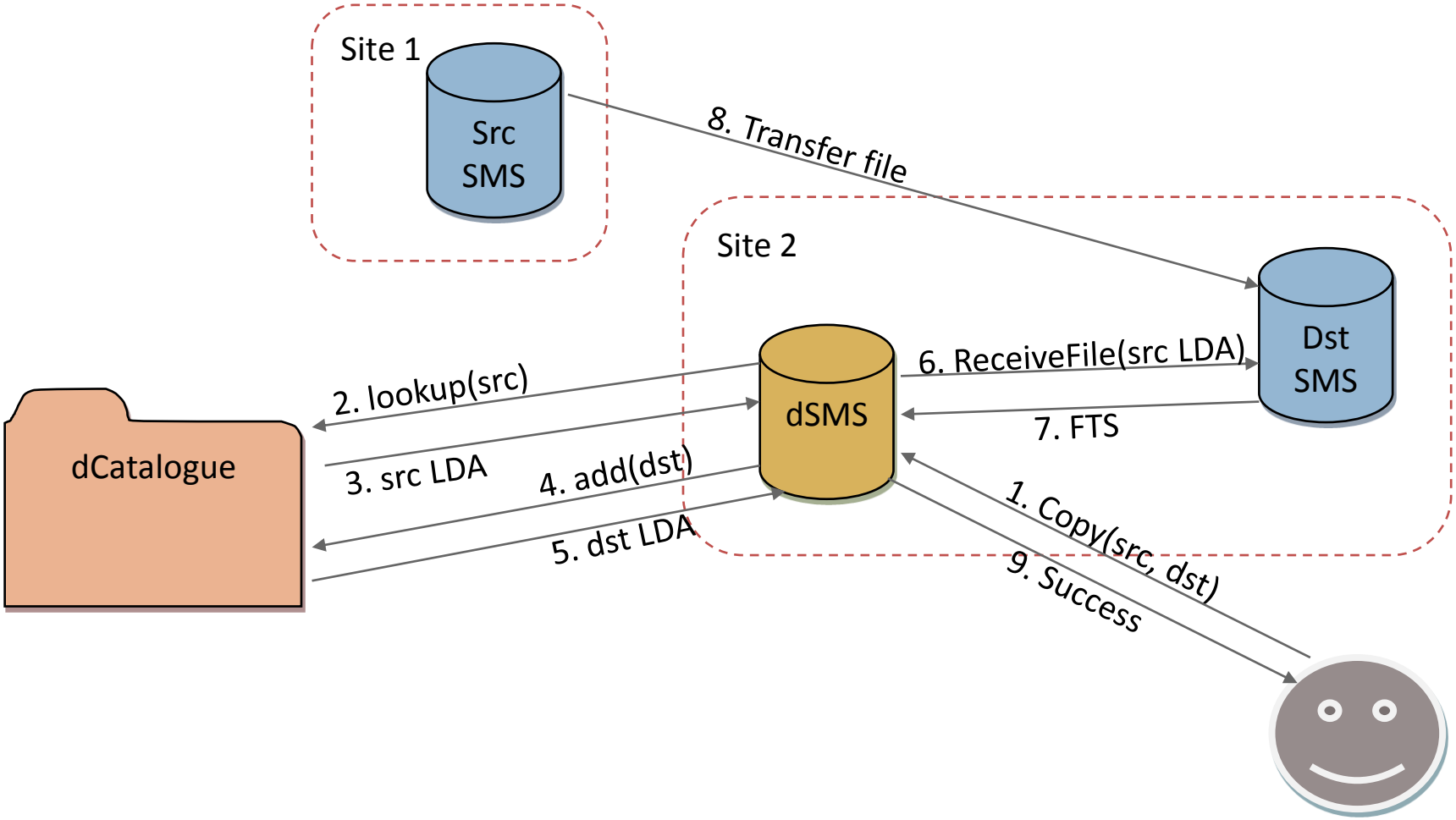
ImportFile operation in dSMS



ListDirectory operation in dSMS



Copy operation in dSMS



dCatalogue

- Central component
- Maps logical filenames to physical locations
- Chooses place to store the new file
- Stores ownership information and authorizes requests

Security

- Clients connects directly only to the dSMS (and not to the dCatalogue)
- dSMS uses trust delegation during connection to:
  - dCatalogue, so user can get information only about his files
  - SMS, so physical files on the target system are actually owned by the user
- If malicious user omits the dSMS and connects directly to the dCatalogue or SMS then he will have access only to his own files.

How dCatalogue chooses destination storage?

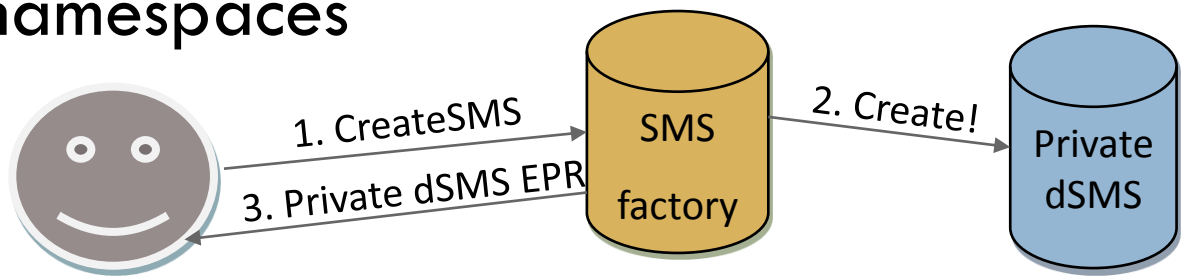
- There is only one algorithm now: round-robin
- Adding new algorithms is easy – you have to implement only one method:

```
StorageEntry findSms(LogicalFilename lfn,
  List<StorageEntry> storages, Map<String,
  StorageEntry> storageByAddress) throws
  NoSmsFault;
```

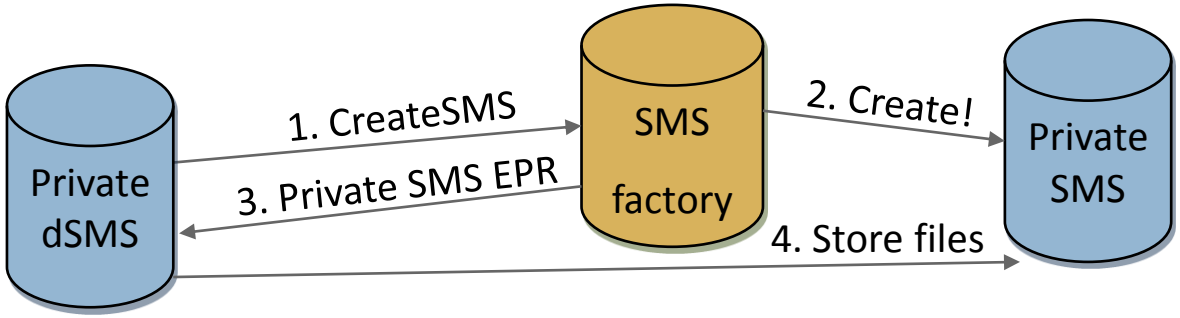
- New algoritms would be useful
  - Eg. storing files from one directory on the same SMS
  - Or use information about free space, server load, etc.

Creating dSMS in Storage Factory

- Storage Factory can create any type of SMS
- It can create also ”private” dSMSes with separate file namespaces



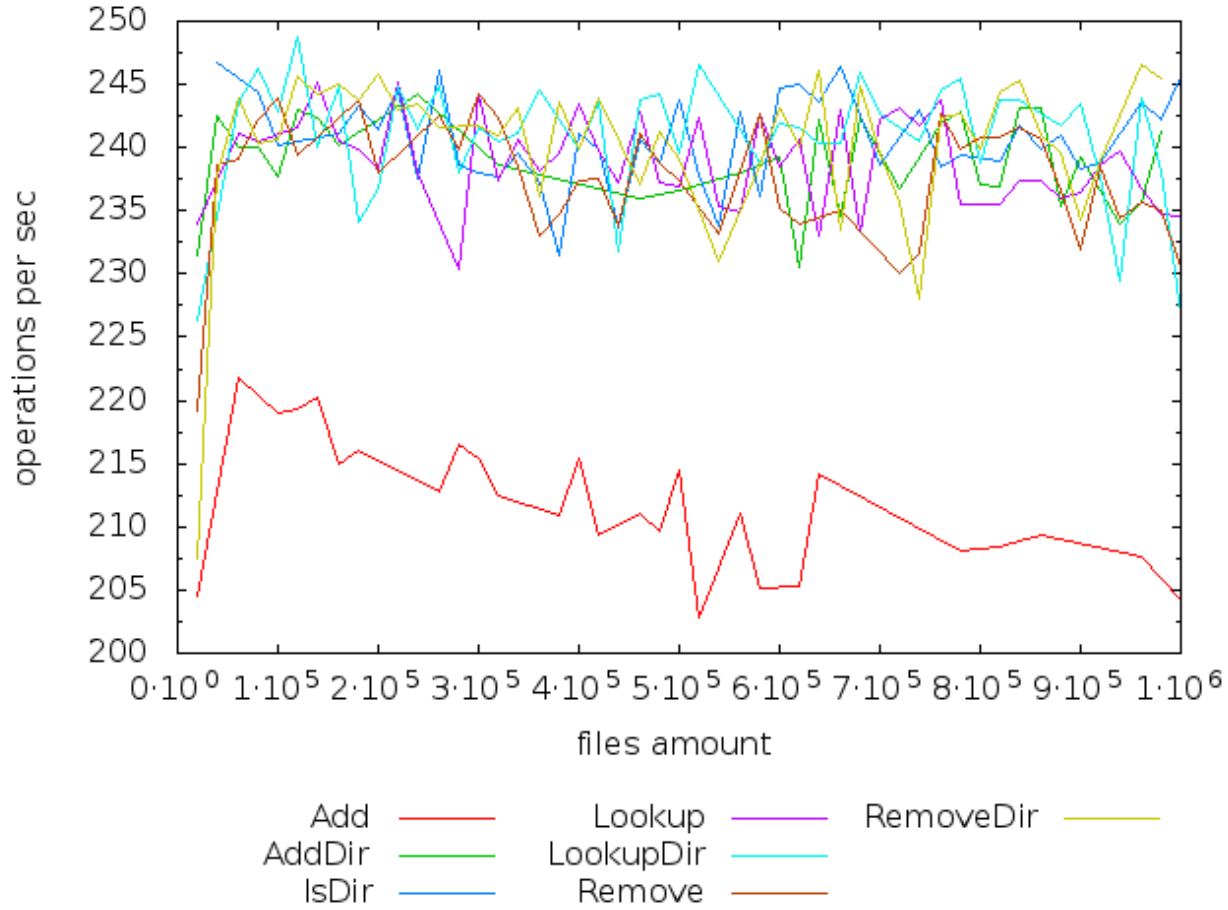
- Private dSMS can also store files in it’s own SMS-es created also with Storage Factory



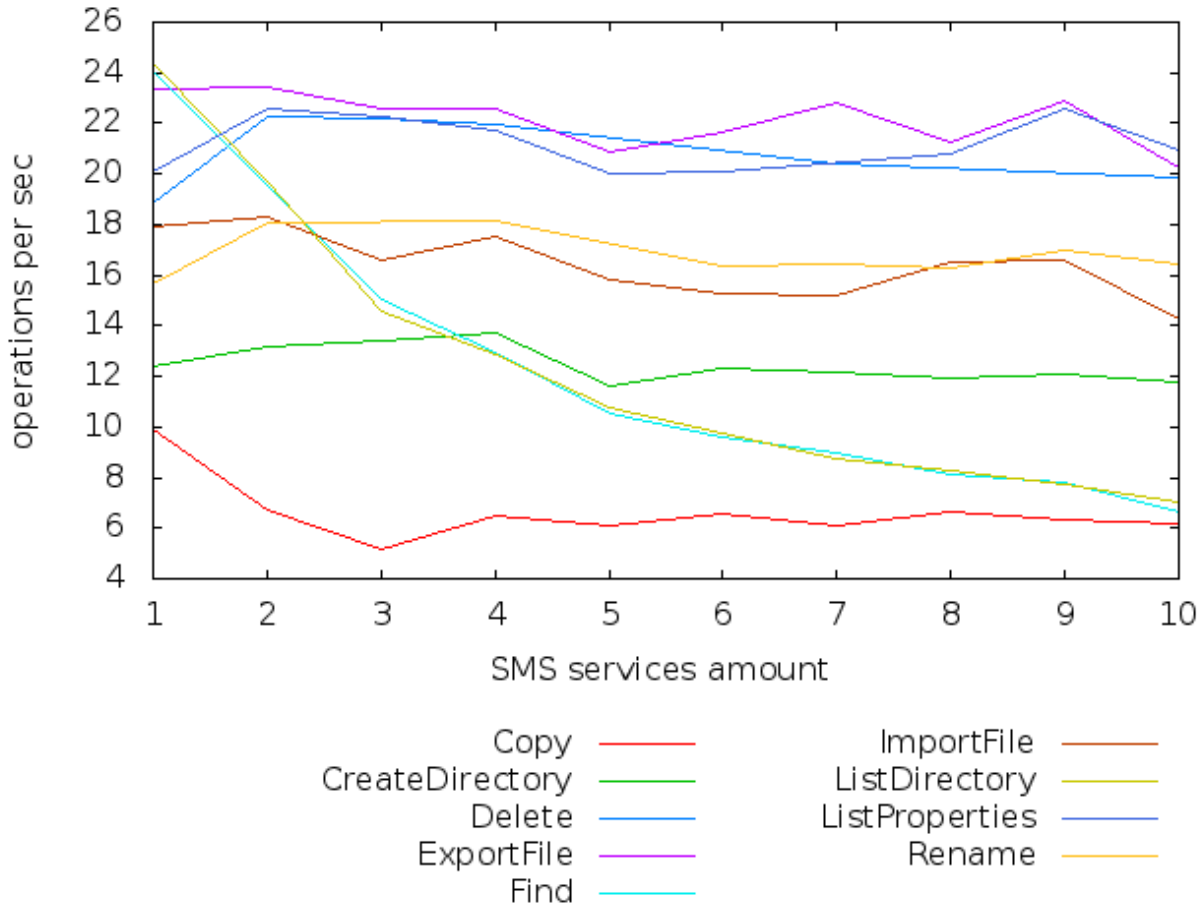
Synchronization (experimental)

- dSMS ensures basic synchronization between SMSes and dCatalogue
- If a dSMS file is removed from SMS (but not from dCatalogue) it'll be automatically deleted from dCatalogue database too
- If there is a new file in SMS it'll be automatically added to the dCatalogue
- Synchronization is done automatically during users requests (which may slow them down)
- Synchronization can be turned off in dSMS config

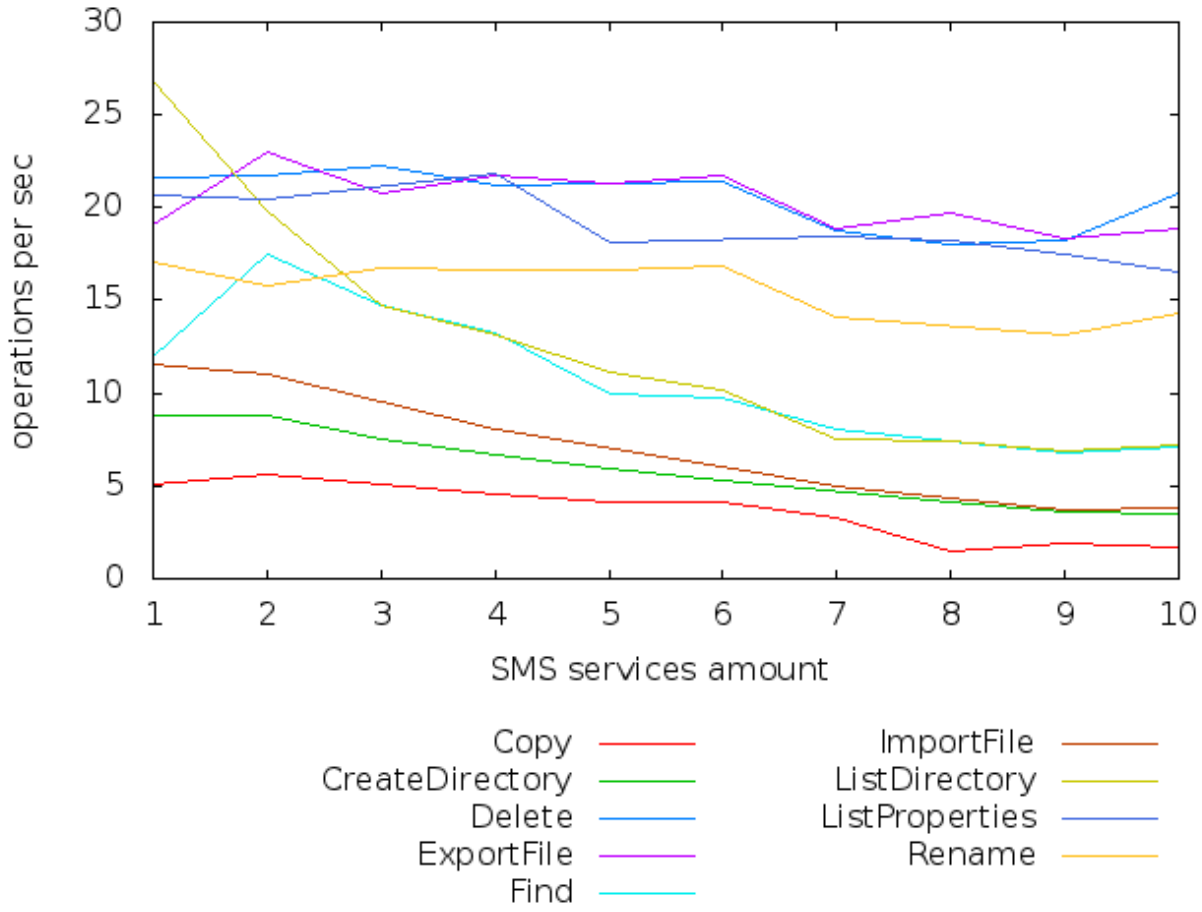
dCatalogue performance



dSMS performance (sync off)



dSMS performance (sync on)



Future work

- New algorithms for storage choice
- Files replication
  - So when one storage goes down other file replicas will be accessible.
- dCatalogue replication
  - Because right now when dCatalogue is broken down the whole dSMS system won't work.
  - Maybe replication at the level of the MySQL database?
- Tools for advanced user
  - Eg. manual choosing file destination.
- New operations for work on a group of files
  - Eg. Calling ChangePermissions method for all files in a directory instead of invoking it for each file separately.

Thank you!

Any questions?