Joint DM-SM TEG workshop

[Day 1 - January 24th - Notes by *D. Bonacorsi* - v.1.0 - last update February 5th]

Federations (Andrew)

Main Agenda: https://indico.cern.ch/conferenceDisplay.py?confId=165687
This talk: https://indico.cern.ch/getFile.py/access?sessionId=1&resId=3&materialId=0&confId=165687

Note: in the following "*{C,Q}*/***Name***" refers to a *Comment* or *Question* by *Name*. In case the person who gave the {C/Q} is not clearly identifiable as from my notes, the field *Name* is set to *Unknown* until the person comments and gets identified.

Introduction

Andrew sets the stage for a discussion about federating SEs. Most experiments have expressed interests and/or also done quite some work on federating SEs already, actually all apart from LHCb which has not much interest, Andrew claims. ALICE is already doing production-wide scale federated storage. US-ATLAS and US-CMS have test or production federations. CMS is further along. This introduction will try to frame the discussion, also based on responses to TEG questions with the experiment feedback.

First, we need to discuss strategies for data federations. Specifically, in the context of the WLCG, key questions are:

✓ what does talking about a federation of storage actually mean in a WLCG context?
✓ how do on-demand / caching architectures (c.f. ARC or Xrootd) fit into the larger WLCG data management ecosystem?
✓ Connecting to Dirk's introduction, there is a huge overlap between SM and DM also in this context, so what are the implications?
✓ How would data federations and federated SEs be managed?

Definition

So, first of all, we have to acknowledge that we do not have everyone on the same page when we say "federation", while we need a common point of discussion ad a "workable definition". Andrew's suggestion in talking about federated storage is to use the definition developed by consensus at the "Federated Storage workshop" [FIXME] in IN2P3/Lyon on November 21st-22nd, 2011 (more info here), i.e.: "*a collection of disparate storage resources managed by co-operating but independent administrative domains transparently accessible via a common name space*". By this, we can now fully set the initial context, by covering (in the following): (a) the common data access models, (b) the storage element hierarchy, (c) the top four concerns (from the initial questionnaire).

Common data access models

We can do:

- Popular Data Pre-placement
- Dynamic Data Placement
- Cached Data
- Direct WAN Access

How do federated strategies address these models (and fit into the larger WLCG data management ecosystem)?

- *Discussion*. Q/**Brian**: do we have, for the dynamic data placement and the cached data, examples of who is using them? Q/**Miron**: even more, what's the difference? why don't we have just cached data? A/**Andrew**: no, because cached data starts with the assumption it's not going to be there permanently. Q/**Miron**: so, any other bullet is there with the assumption the data is going to be there permanently? what's "permanently"? C/**Andrew**: "permanently" means that for example ATLAS data is going to be on a SE until ATLAS remove it. C/**Miron**: ok, so, for all this discussion the point is who makes the decision to manage it. We should just talk of cached data, because all the data that you put there is not single-copy, not a disaster if it disappear, and the only point is deciding who decides to remove it. C/**Dirk**: with e.g. CERN cached data, you have to live with this, and you have to annotate which level of caching is used. C/**Miron**: show me in the system all the places that you booked data caching: I do not think that by separating it you gain anything, but I would argue you loose. C/**Dirk**: that's another discussion: for the sake of describing which cache you are talking about you may be right that there's too much caching, that it does not really help, that we should not do it in five levels, etc.. C/**Andrew**: as part of the discussion, we can simplify the scenario. C/**Unknown**: there are two kinds of DM caching: one is an external entity that takes care of freeing up space (i.e. the experiment), and there is another one, an internal entity entity, the storage system that says it is full, it needs space and it automatically purges. C/**Ian**: there is an advantage in talking about caches. If you call it cache, you are forced to think of it in a particular way. And whether is that you clean this, or whether something cleans it for you, or whether it cleans itself because a file was inaccessible (e.g. a disk crash).. but the advantage to call it a cache is that it forces you to think of it as an *inconsistent resource*. For the standpoint of moving forwards, since we know it is an inconsistent resources, maybe we can call it a "*(un)managed cache*". But it's an advantage to have this mindset change: you have a disk resource, which is a non-permanent resource, which we treat as a permanent resource right now. C/**Dirk**: agreed. I do not like when people talk about a cache that is exposing its name space and stuff like that. We should probably let this drop and disappear, or do not rely on them anymore at least, but we also should annotate whatever level we are currently caching on it, similar to metadata when one talks about it and people do not understand which metadata one means. C/**Andrew**: what about letting disappear "Cached Data" from the slide? C/**Miron**: not really. What I would like to see is "Eager", "Lazy" and "Just-in-time" caching. Q/**Miron**: I have another question (from a OSG blueprint): when you talk about access, do the users expect data to be stored at sites under their local user ID? A/**Andrew**: it could be, but if you look at the experiment answers, this never really came up as a feedback. Probably they think of it, but never verbalize it.

SE hierarchy

The other thing when we talk about federations is the SE hierarchy. Every experiment has a slightly different one, Bu minimally, we can say we have the following:

✓ Tier 0 – Top most central repository;
✓ Tier 1 – Next-in-line repositories;
✓ Tier 2 – Medium-sized "regional" repositories;
✓ Tier 3 – End point, typically local end-point locations. Mainly used for local analysis.

Where does a federated strategy make sense? And fit into the larger WLCG data management ecosystem? Should it pervade the whole hierarchy, or limit to some level of it?

- *Discussion*. Q/**Miron**: can I have Tier-4? It's some storage that I got for the next 24 hours (call it cloud, don't care). This does not limit by thinking that the storage is all something that belongs to me. What do I do if somebody gives me half a PB for 2 weeks? C/**Andrew**: we can certainly add it. C/**Brian**: one way to look at this is the amount of controlled vs unmanaged. The T0 is 100% managed. Others don't. The T3 are 0% centrally managed, and 100% locally managed. C/**Dirk**: but this rolls back to the caches discussion, if you would properly label everything as cache you will not understand which service level at the Tier would match with the responsibility the Tier has. C/**Miron**: I understand that T0 and T1 both give you caches and archives, and are hence make them different from the others, but these are two different storage spaces. C/**Paul**: some T2s also provide archival storage. C/**Dirk**: but do the experiments use it? C/**Paul**: as far as I know, yes. C/**Miron**: but do they count on it as storage? C/**Ian**: maybe for ATLAS, but for the case of CMS, the only custodial data (it's labelled as such and treated differently in the DM system) reside on T1 resources.

Top four concerns

The top four concerns are:

✓ Too much complexity
- DataManagement
- Architectures

✓ Lack of Robustness
- Transparent file access failover
- Too many single points of failure

✓ Performance & Scalability
- Difficult or complicated to scale present systems

✓ Interoperability

The question is: Do federated strategies make these better or worse? And how would they fit into the larger WLCG data management ecosystem?

Our endpoint

We need to come up with a recommendation. There are really only four possibilities:

1. Develop something new to federate storage
2. Encourage existing technologies
   - We can recommend which ones
3. Be passive
   - Just wait and see where things go (e.g. from the market?)
4. Do nothing
   - This may not be a viable recommendation. Experiments will in fact do something.

- *Discussion*. Q/**Miron**: can you have the Data Placement problem without having the Federation? A/**Andrew**: they are orthogonal questions. C/**Miron**: when we talk about data placement we talk about "where". Talking about federations we talk about groups of data sites. Let's assume that we remove the "common name space" to the definition of Federation, if so probably the Data Placement fits the definition and is applicable here, so why should I have a problem with this? C/**Wahid**: the "transparently accessible" also should still apply. C/**Miron**: the whole idea of data placement is that you can access the data transparently regardless of where it is. C/**Wahid**: not in the storage case. You can access the data wherever it is but with the workload management sending your jobs where the data is. C/**Andrew**: the point is that you do not need the workload management to access the data, that's where the transparency is. C/**Unknown**: so we are not supporting anymore any concept of data locality? C/**Andrew**: you want to access the data wherever they happen to be, so data placement becomes absolutely immaterial. I am claiming the Data Placement and the Federations are two completely separate problems, because you have to solve the data placement problem regardless of whether you have a federation or not. C/**Miron**: data placement means that you have independent, locally autonomous sites, and this is the key, when you have to access them most of the hard issues we will face are there. If you have a data placement problem you have most of the hard of the federations. The name space is not the hard part, just someone has to know where data is, so you create some kind of a common name space, so it's well visible, and you will quickly realize the gap among the two is small, but let's keep things going right now. C/**Dirk**: yes, there is probably no other way otherwise for the system to transparently pick the replicas, so the federations must be global. C/**Philippe**: we need to agree here on what is the problem we are trying to solve here. Do we still want to have some job brokering system that takes into account the location where we think the data is and to have the federation as a backup in case the file is temporarily or permanently unaccessible - for which we indeed need a common name space - or we want a completely chaotic system in which the data is pulled on some SE for some time with cache life time defined somehow but transparently to the user? If we are talking about the former, LHCb subscribes to this and we need a common name space (like the PFN, GUID in some sense as such) and then we could have applications that are smart enough to look up a file catalogue and find another replica and get another TURL from SRM or whatever, and e.g. GFAL would be able to do that. What do we aim to do here? C/**Dirk**: at the workshop (LHCb not absent) this was discussed, and we talked about jobs being killed if the jobs cannot find the file locally and cannot read it remotely... C/**Philippe**: fine, but then you need to put some intelligence at the level of your applications, which must be smart enough to first lookup the file locally and only go outside in case this fails. C/**Andrew**: the only real agreement we had in Lyon is that eventually the job is brokered where the data should be, and in case the data is not there or not accessible that's

where the federation comes in, and you can access the data from somewhere else. We did not go further on what happens at that point: you access on WAN, you fix the replica catalog, etc and all experiments had different views. C/**Miron**: we also did not because this was driving big site issues. Caching is a small site issues, in respect. C/**Andrew**: from this standpoint, federating sites becomes a somewhat easier because that's a manageable problem. C/**Jeff**: the ability of the experiments to predict where the data should be is essentially zero. E.g. in predicting the placement, ATLAS succeeds at the level of 10%. C/**Simone**: 15%. C/**Jeff**: ok, so 85% of the data movement is for nothing. We should have gone for content-delivery networks? C/**Miron**: we want to have a framework that allows you to fix the weak points. It was a mistake to go down and do a lot of data placement and never do remote IO. In my little shop we have been doing remote-IO since 1982. Federation is just a buzzword, what you want to do is remote IO: the data I am looking for is not there, I want to find it and access it. I want an architecture in which you can decide how much you do data placement and how much you do remote IO, i.e. how much "eager", how much "lazy", and so on. "Eager" is when you do it at the very beginning, "lazy" is when you send a job (let me move the data before I send a bunch of jobs to access it). Every system will be doing some files in a pre-staging mode. Do not put yourself in a corner, it's either this or the other one, because your workload may change, technology may change, the knowledge of your system may change. C/**Jeff**: if you know how to predict you won't be able to do it, but if you don't know how to predict you don't want to have to care. C/**Miron**: if everyone is using remote IO you need to have safeguards that avoid the system to crash. We know that if you are not using locality of reference, the system will not work, because you don't have the bandwidth to deliver all these things all the time, so there must be some locality of reference, but how much you can accomplish it depends. C/**Daniele**: Can we agree that these two worlds can coexist and the recommendation would be to analyze and spot among coherent architectures the most promising ones? C/**Unknown**: we should find what the gap is with respect to what we have today; is it just the naming of the file? C/**Jeff**, **Brian**: this is an important decision to be made. C/**Miron**: you need an architecture in which any client/server can be a server/client because you want to have recursion in the system, you want to be able to use it like a router, to have answers like "I am the address you looked for" or "no, I am not, but I know how to get you to the address". Every client can act as a server at the next layer. C/**Dirk**: like xroot. C/**Miron**: we had this issue in the gatekeepers while dealing with the proxy delegation, due to the fact we were always thinking in one hop, but we never have just one hop in our systems. C/**Wahid**: we have some fault-tolerance in our systems in the sense you resubmit the job it goes to another sites and it does not fail there, but you probably need to consider you have to fix the catalogue. C/**Dirk**: there are several possibilities, here. The job can realize it cannot get the file from here, but the same job will go to another storage and get the file from there without copying it. The other configuration - discussed at the Lyon workshop - was that once you figure out the file is not actually there while the catalogue states it should be there, rather than go remote and just read, you copy the file locally and basically re-instantiate the consistency of that catalogue entry. C/**Miron**: the question is who detects the failure? and what to do with such knowledge? is the job kicked out? put on-hold? and another question is: how do you fix the failure? e.g. is it caused by an inconsistency in the catalogue? and remember that if you want to bring in a file, you have to find room for it, so if you do not have space what are you going to throw away? if you have space of course you keep all the replicas and you are done, but if if you don't have space, how do you decide how to make the needed space? C/**Dirk**: agreed. trying to summarize, going remote is one possible solution to get back some jobs that would otherwise fail, but that does not give an overall solution. C/**Andrew**: an experiment would tell us that if a job was brokered here, it happened because the broker looked into a catalogue, and the catalogue said the file was there. If the file turns out not to be there, then the catalogue is wrong, so a decision must be taken... C/**Miron**: I don't care about what the decision is because we should be able to change the policy on the fly. I care about who makes the decision. Is it the policy engine running on the site? is it something that needs to escalate somewhere else? and you need to decide what to do with the job. C/**Philippe**: I can tell you about what DIRAC does for LHCb. Neither deleting the file from the catalogue nor replicating the file are good. If you remove the file from the catalogue and it was just a server down which eventually comes up, the file is still there, it is back available while its entry in the catalogue is lost. If you replicate the file it's probably not so bad, but as a result you end up with 2 copies of the file when the server comes back and then you have dark data on the storage. The way LHCb/DIRAC does it is reporting to a DB that contains info about problems occurring at a given point for that specific file. The replica is flagged as not-available in the catalogue. The key point is how to re-activate this replica and solve all problems: for this, there is a human being who makes the decision and fix. C/**Miron**: what you have to capture in your description is where you put this policy. There must be a clear place where this policy should be expressed and managed, so if tomorrow morning someone wants to change it it is possible. The question is: is this delegated to the VO? does the site do this? is it a service provided by a third party? This policy piece of the decision has to be structured. Also, an additional problem is the storage space. Who decides what to

throw again? C/**Philippe**: this is a second order problem, when you have this problem and in addition a lack of storage cspace. Q/**Miron**: but if you have space why don't you replicate all data? A/**Philippe**: we have space, not all that space... for one file is fine (~5 GB). C/**Paul**: you can think of an architecture in which you have a partition of the available space dedicated to this recovery procedure. C/**Miron**: how big should the space be? how many server should not do down at the same time? and what do you do when you are running out of space? These decision has to be articulated. In most cases, this decision is either not taken, or discovered when there is a disaster that this was not thought of carefully. And that's the main deficiency of our infrastructure: when something goes wrong, the system can go very wrong. C/**Paul**: I would concentrate on the place itself where the policy should be, more than the policy itself. C/**Simone**: you are discussing a scenario in which for some reason a file you needed is not there. This is of the order or 1/1000 of the failures we see today. The biggest problem in data access is not what is being discussed, since the files don't automatically disappear. The main problem is that when a SE on a site is overloaded it does not serve any file. What do you want to do? do you want to get them all of them from another site? Then you overload another site. What happens is instead only one server is overloaded? Then you get e.g. 9 files but you cannot get the 10th. You read remotely only the 10th? You copy it locally? You could overload it further. Or fill the local disk. C/**Dirk**: But you are talking here about the repair. C/**Simone**: if we want to talk about a repair system, you need to get back to the experiment land, because otherwise you do not have the needed info. C/**Miron**: we are not talking about a use-case, we are talking about decisions that need to be made. You want data, and the data is not locally available. What do you do? Who makes the decision? Where is the decision made? E.g. the decision should not be different if it is for 1 byte, or 1 Gbyte. The question of the catalogue is still a question of who is managing the name space. There are known solutions, in which each file has a own directory at a given site, and it does not mean the file is there but it means the file is associated with it. So we only need to state that this decision has to be made, and these are the places where it can be made. C/**Simone**: we should take into account many scenarios, including the disaster loss of many files. See Philippe: human intervention needed, room for automation, etc. C/**Dirk**: the automation can only happen in the sense of automating the failover. C/**Unknown**: we need to consider the automation also in the repair, we have a system that is labour-intensive in terms of monitoring, sites, etc. C/**Dirk**: you can only automate after you understood which the frequency of each failure occurrence. We need to be careful on what can be done automatically. C/**Brian**: so, the intelligence and the policy decision should be at the experiment level. C/**Miron**: what is a catalogue? it is an implementation of assumptions, of a function give me some info, I give you some more, how to go from one place to another. C/**Andrew**: we should try to keep this a high enough level to avoid talking about specific implementations C/**Brian**: These are TEGs, and the T stands for Technology., we were asked in the mandate to be concrete and we should at some point talk about specific technologies. C/**Dirk**: concretely, we should list the key concepts and then eventually go to technologies, not the opposite (e.g. to just pick an existing one and that's it). For some things that Miron quoted e.g. we do not have a technology at the moment, e.g. the policy escalation chain, this is something to be looked for in the future. C/**Wahid**: we can add components, but most of the things should roughly exist already. C/**Unknown**: are we talking about an evolutionary or revolutionary process? C/**Wahid**: evolution, mostly. You should not be afraid to break anything. C/**Miron**: it would be an evolution for this community to sit down and think about it in terms of structure and principles rather than technology evolution. This would be a revolution in itself, and once this is accomplished, we should follow an evolutionary process to translate this revolution into something real. Otherwise, one risk is that a discussion on evolutions will immediately deteriorate in discussion on technology, like one team is doing in one way, we will keep doing it like this, we will tweak here and there, and there will never be a common ground to talk about it. C/**Brian**: we should work on how federations work out there in various forms using different protocols, but not exclusively recommend one or the other. C/**Unknown**: we should really start from the gap analysis. C/**Markus**: this report is potentially useful or potentially extremely dangerous. We should profit of the window of opportunity we have with the long shutdown in front of us, and not loose it. We should do both, concepts and implementations: we should write down how the ideal system would look, its qualities, and do a kind of strawman roadmap, and also naming existing technologies, saying that these has to be there and used on a certain scale, and move very quickly to expanding the scope of using this, so we can get experience that we can't achieve by just thinking. C/**Dirk**: agreed, it would be good to split the concept, theoretical part and the pan of actions where the technologies are named. C/**Daniele**: The gap analysis can be the first step of this. C/**Brian**: agreed, that is the first part, then the rest as Dirk said.