Spade: Decentralized Orchestration of Data Movement and Warehousing for Physics Experiments

Simon Patton*, Taghrid Samak†, Craig E. Tull*, Cindy Mackenzie‡
*Lawrence Berkeley National Laboratory,
†Google Inc.,
‡Formerly with University of Wisconsin, Madison

***Abstract*—The paper describes the implementation and experiences gathered from the Spade project which is a decentralized mechanism for data management that has been used in High Energy and Light Source physics experiments. The background and motivations that shaped the Spade project are covered, followed by an outline of the current design of the deployed application. The operational scenarios that have been developed from the use of Spade in a number of difference contexts are enumerated, after which there is a short discussion on how the implementation of the application has progressed. The paper finishes with some highlight of Spade's performance and a conclusion about what has been learnt from this project.**

I. Introduction

As more and more science experiments collect larger and larger datasets, it is harder for scientists to analyze their data in-situ. To be able to extract useful information from this ever-growing pool of data it is necessary to move these massive datasets to computing centers that have enough processing power to analysis and archive them. The Spade application was designed to address this critical need. In order to be useful to scientists it was also required to use minimal infrastructure while providing robust data transfer, especially when the network conditions cannot be controlled or the networking resources are very limited.

Spade was first used by the IceCube experiment to move its data files from the South Pole to Madison, Wisconsin. The IceCube experiment [1] is a neutrino observatory that uses an array of light detectors buried in a 1 km^3 of ice to detect the Cherenkov radiation emitted by muons produced from neutrino interactions. The challenge here was that the network connection between the South Pole and Madison was only available for around 12 hours a day due to the orbits of the communication satellites. This required a system that was robust to a complete loss of network. The satellites also limited the amount of bandwidth that IceCube could use, initially to 4GB/day, which meant that any application would need to track the progress of transfers and cache all files that could not be transferred, either due to loss of network or bandwidth limit, and attempt to re-transfer them at a later time. Moreover the Office of Polar Programs required all data to be accompanied by metadata conforming to the GCMD [2] in order for it to be included in the Antarctic Master Directory [3] so Spade was also required to manage the metadata of each transfer.

Another high-energy physics experiment that had similar requirements to the IceCube, but required support of multi-destination and more generic metadata was the DayaBay Neutrino Experiment [4]. This experiment required the data management infrastructure to move its data files from the experiment's location, just outside Hong Kong, to computer centers both at Beijing and at Berkeley. Using Spade as a solution for this experiment brought new challenges such as; separating out the experiment-dependent features from the generic ones, dealing with data rates that at least 50 times higher than the IceCube case, and sending data to two destinations rather than one.

Spade's success in dealing with the data management issue for IceCube and Dayabay has lead to it being incorporated in the new SPOT suite of tools that are being developed for light sources [5]. In the context of this suite, data is being transferred from the Advanced lightsource (ALS), a Basic Energy Sciences (BES) X-ray synchrotron facility at Lawrence Berkeley National Laboratory (LBNL), to the National Energy Research Scientific Computing center (NERSC). Beamlines at the ALS generate terabytes of raw and derived data each day and serve 1,000's of researchers each year. Yet, much of the data remains unanalyzed due to barriers of data management and processing that smaller science teams are ill-equipped to surmount. In the past 2 years, ALS scientists have seen their data volumes grow from 65 TB/year to 312 TB/year, and in another 1-2 years will be generating 1.9 PB/year. In addition to the ALS, the SPOT suite is working with other light sources throughout the USA, such as the LCLS at SLAC, the APS at ANL and the NCLS at BNL.

The addition of light source experiments to Spade's portfolio has, again, given rise to new challenges. In this case datasets are composed of multiple images taken in a very short space of time that should be treated as a single datum so Spade must evolve from handling single files and their metadata to a generic system where one or more files are collected into a 'bundle' along with its associated metadata. Also, as light sources are user facilities, Spade needs to address the ownership of files in the warehouse (previously they were usually all stored under a collaboration psuedo-user account.)

We begin by a high-level overview of Spade design in Section II. Different deployment scenarios are explained in Section III, followed by the implementation details in Section IV. Results from recent experiments are presented in Section V, and conclusion and future directions in Section VI.

II. Spade Design

The design of Spade centers around three main stages of operations: local domain operations (source); data transfers; and remote domain operations (destination). Each deployment of Spade can support all three stages and each "source" can have more than one "destination" so it is possible to create a network of Spade instances in order to manage data distribution for a widely dispersed collaboration. The following few sections describe the details of each stage.

A. Local Domain Operations

Figure 1 shows an overview of the operations that Spade executes within the local domain, in other words what happens at the source of the data. Operations begin with the *registation* of a local file stream. This registration takes the form of a file that describes, among other things, a directory and file pattern for Spade to use to detect *semaphore files*. A semaphore file is used to indicate that there is a bundle ready for Spade to manage. As Spade has developed, the relationship between a semaphore file and its associated bundle's location has grown richer, suffice it to say that, given a semaphore file, Spade can then locate a file or directory that contains the data for the bundle.

The data, along with its associated semaphore file, are fetched from their locations by the *Fetcher* task and placed in Spade's cache. The main aim of moving the files is to protect the data from being accidentally overwritten or deleted before it has been successfully transferred. However, in a typical Spade deployment, SPADE will be executing on a Data Transfer Node (DTN) that is separate from the node running the Data Acquisition (DAQ) and so moving the files into Spade's cache also frees up space on the DAQ node for more data to be taken.

Once a bundle's data and semaphore file are in the cache they are examined by the *Examiner* task that is responsible for creating the metadata associated with the bundle and also for deciding how the bundle will be handled by the rest of Spade. Metadata that is unique to a bundle is normally contained in its associated semaphore file, although this file may also be empty and metadata, such as the file creation time, be extracted from elsewhere. Metadata that is common to all bundles in a file stream can be specified by the registration with which the semaphore file was detected. Also contained in this registration is a set of remote Spade instances to which the bundle should be transferred along with a set of flags to specify what should happen to the bundle locally. The Examiner task copies these outbound destinations and flags into the bundle's *transfer ticket* that is then used by Spade to shepherd the bundle through the rest of its local operations.

Many experiments want a fast check of their data that is being taken in order to, among other things, monitor its quality and decide if experimental conditions need to be changed. For that reason Spade has the option to duplicate bundles onto local storage at the experiment. This duplication is done by the *Duplicator* task. Once a bundle has been duplicated Spade has no further responsibility for that copy of the data. It is up to an external system to detect the arrival of the data (signalled by another semaphore file) and make appropriate use of it. It is also not Spades responsibility to delete the duplicated bundle so some external system must monitor that storage space.

For experiments that want a more organized approach to data local to the experiment, the bundle can be placed in a local warehouse using the *Placer* task. While this activity used to be the sole purview of the remote domain operations when, during its evolution, all activities were supported by all Spade instances, this activity became available to local domains too. Therefore, while it is mentioned here it will be dealt with in detail in section II-C.

Similarly, analysis of a bundle by the *Analyzer* task - not to be confused the use of a duplicated bundle, which is not under Spade's control - is mainly geared to execution at the remote domain so is dealt with in section II-C.

B. Data Transfers

In order to keep data transfers manageable and efficient, bundles are transferred as single files. The first step in this is to wrap up the data and metadata into a single file using the *Wrapper* task, which takes all of the relevant files and builds a single, wrapped file. Currently data and metadata files can be simply 'tarred' up into a single archive or a richer wrapped file can be created containing the bundle's information by means of using the HDF5 format.

While the wrapped file is sufficient for transfer, there can be a benefit in compressing (and uncompressing) the wrapped file as this can reduce the bandwidth transferred at the cost of time spent compressing the file. In the case where a bundle is transferred more than once, either to two remote Spades from the local one or by relaying to a third Spade via the remote one, the compression time can be amortized over the number of transfers. For each file stream the registration states whether it should be compressed or not, and if compression is required it is handled by the *Compressor* task. Currently both 'gzip' and 'bzip' compressions are supported.

When a bundle is ready for transfer its file is referred to as a packaged file. The transfer of a packaged file is managed by a set of *shipping tickets*. These are identical to the bundle's transfer ticket, but with each instance having just a single outbound destination. The array of shipping tickets, one for each destination specified in the bundle's original registration, is used by the *Shipping* task, which is a multiple-instance task with one instance for each shipping ticket, to transfer the packaged bundle file to a receiving directory at the appropriate remote Spades. Any failed transfers are queued to be tried again later and this is repeated until the transfer succeeds. Once a packaged file has been successfully transferred to a remote Spade the Shipping task transfers an empty file to act as a semaphore file for the remote Spade.

C. Remote Domain Operations

Figure 2 shows an overview of the operations that Spade executes within the remote domain, in other words what

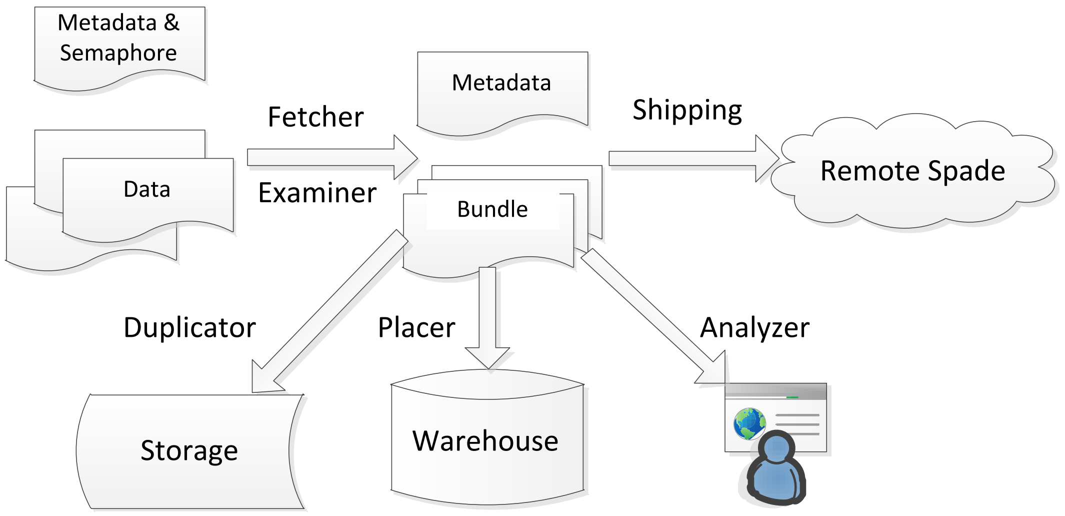

Figure 1: Spade operations within the local domain.

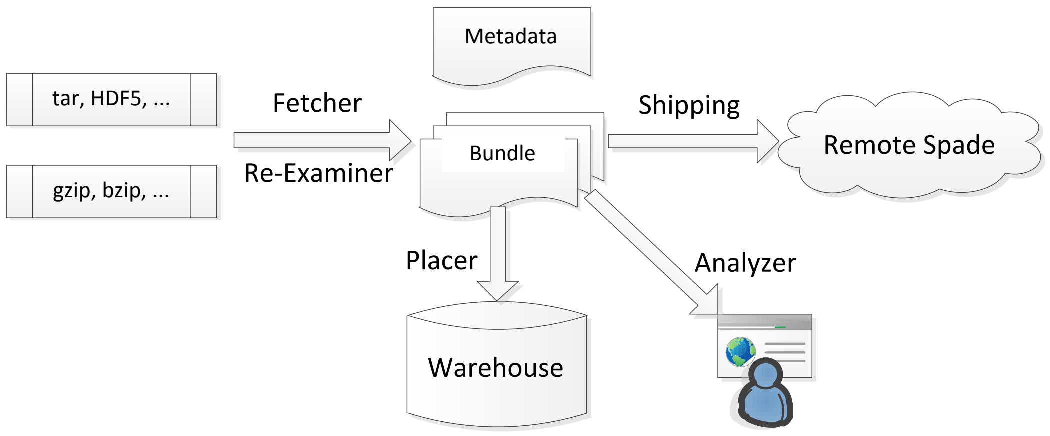

Figure 2: Spade operations within the remote domain.

happens at a Spade instance that receives data sent from the local one. As with the local domain the presence of a bundle that needs to be handled is signaled by the existence of a semaphore file. In this case it is the semaphore file written by the local Spade's Shipping task and it indicates the bundle's transfer is complete. A received bundle may or may not have an explicit inbound registration. If it does not have an explicit one then a default one is used which attempts to warehouse, analyze and archive the inbound bundle. An explicit registration is only needed to override this default behavior. An inbound registration specifies a local Spade and registration identity on that local Spade that must match that of the inbound bundle in order for that bundle to be assigned to that registation.

Once an inbound bundle has been detected it is fetched by a second instance of the *Fetcher* task and placed in the remote Spade's cache. Once there, the bundle is examined by the *Reexaminer* task that is responsible, using any inbound registration assigned to the bundle, for filling the bundle's *transfer ticket* that is used, as with local operations, to shepherd the bundle through the rest of Spade.

Before a bundle can be stored or analyzed it must be unpacked from its shipped file. This normally takes the form of uncompressing the bundle, if appropriate, using the *Expander* task and reconstituting the original file or files using the *Unwrapper* task. However, experience has shown that there are cases where the wrapped file is the preferred storage and analysis format of the data. In that case the Unwrapper task simply extracts the metadata of the bundle and leaves the wrapped file as the bundle's data.

Once a bundle has been unwrapped it can be placed in the data warehouse that Spade manages, unless the inbound registration declares this not to be the case. This is done by the *Placer* task that uses the bundle's metadata to calculate a path, if any, within the warehouse where the various incarnations of the bundle – packed, wrapped and data – are to be stored. This task then tells the warehouse to store the bundle at the calculated locations. As well as storing the bundle in a warehouse, which is expected to be some form of direct access, Spade may also write the bundle to tertiary storage using the

Achiver task. This calculates a storage path for the packaged file, which may or may not be the same as the Placer task's, and then attempts to write it to that location in the tertiary storage system. If the attempt fails it is retried at intervals until it succeeds. Meanwhile the packaged file is kept in the cache until archiving is successful.

If a bundle is stored in the Spade warehouse it is possible to execute an analysis on that bundle once it has been placed. The *Analyzer* task takes care of doing this by calling a specified script and an appropriate set of arguments that depends on the bundle.

An inbound registration may also specify one or more outbound transfers in which case the packaged file is sent to additional remote Spades. These transfers are done under the auspices of the remote Spade and its inbound registration, so the Spades that are receiving the re-transmitted bundle only need to know about their nearest neighbor and do not require any knowledge of the original, local, Spade.

After an inbound bundle has been handled by the remote Spade, the last significant step is to calculate the orignal checksum of the incoming file, using the *Confirmer* task, and send it back to the originating Spade in order to make sure the transfer was successful. This confirmation is only done after the remote Spade has successfully finished with the bundle in order to avoid data loss that could occur if the remote Spade fails somehow and the local Spade has already deleted the transferred file.

III. Deployment Scenarios and Usage

This section describes different scenarios for deploying Spade. Each has its benefit and specific use cases from our experience. For example, our first deployment in IceCube was the basic Remote Warehousing for just moving the data between two locations. More requirements from scientists and also more constraints from the networks dictated our deployment configurations. For example, to be able to look at the data and perform some analysis while the transfer is taking place, the data need to be duplicated locally. The DayaBay requirement for moving data to multiple destinations (Beijing and Berkeley) resulted in two deployment options; Relay and Fan-out. The following subsections explain each deployment.

A. *Remote Warehousing*

Remote Warehousing was the original motivation for Spade. In this scenario, data is moved from its source to a warehouse at a remote destination. Two Spade instances are needed, one at the source and the other at the destination. Figure 3 shows this basic configuration.

A bundle's data is created at the source. The bundle and its metadata are temporarily stored in that Spade's cache and its packaged file is transferred to the remote Spade. At the remote Spade the bundle and its metadata are placed in the warehouse and analysis of the delivered bundle is triggered. The bundle can also be archived onto tertiary media. Confirmation of successful handling of the bundle and its metadata it communicated to the originating Spade so that it can be removed from that Spade's cache.

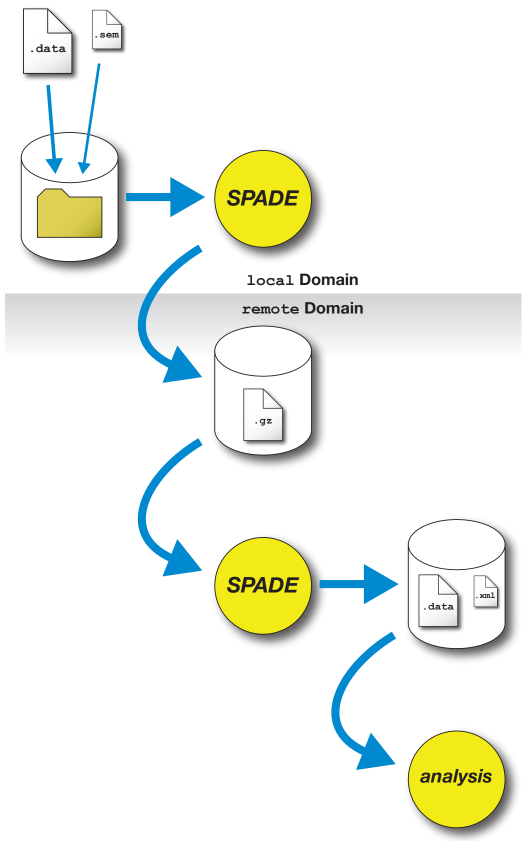

Figure 3: Remote Warehouse: basic Spade deployment for remote data transfers.

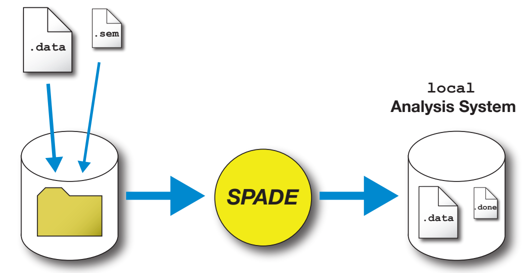

Figure 4: Spade deployment for local duplication

B. *Local Duplication*

It soon became clear that in order to get an immediate assessment of the data, and thus adjust the experimental set up, a local copy of the data would be necessary. Rather than directly access the output of the DAQ, which could significantly impact its ability to record data, Spade was given the ability to send a copy of the data to storage on a local analysis system. Figure 4 shows how Spade does local duplication. The data copy is not managed by Spade leaving the analysis managers to decide on the policies of how to use this instance and when to delete it.

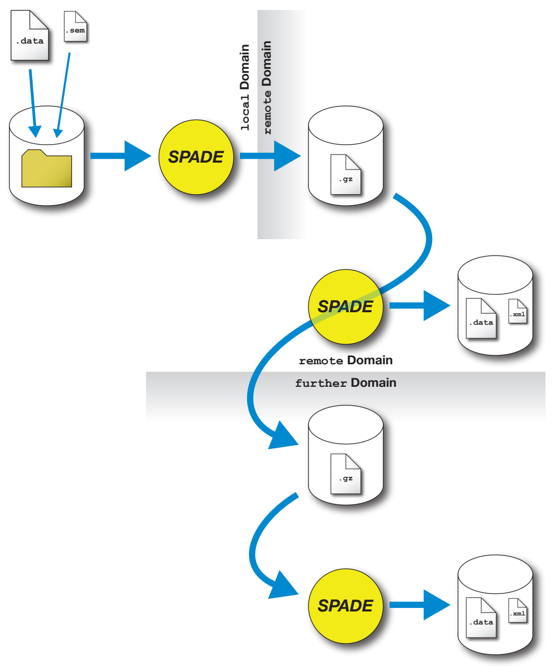

Figure 5: Deployment for multi-destination transfers using relay.

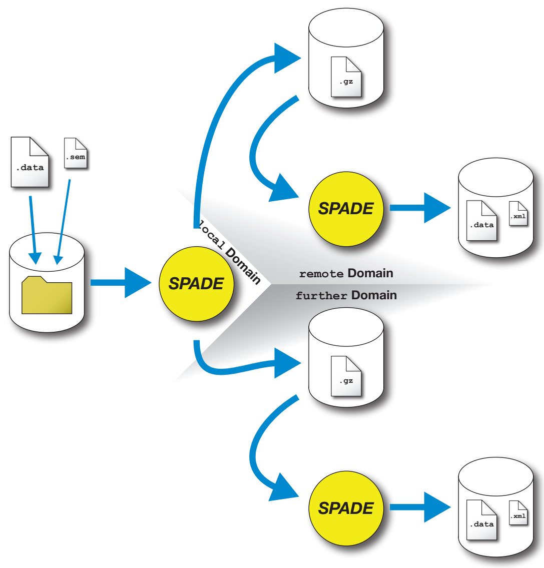

Figure 6: Deployment for multi-destination transfers using parallel processes.

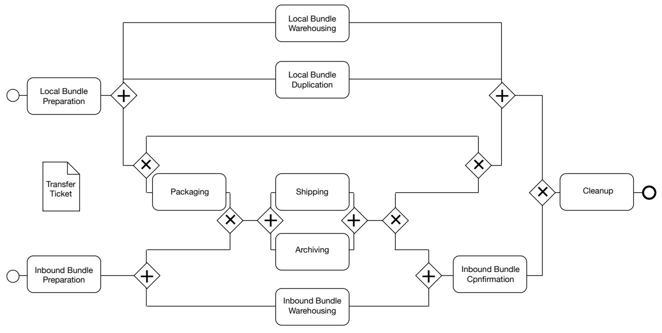

Figure 7: The high level BPMN diagram for Spade's main workflow.

C. Relaying Data

The introduction of Spade to the Dayabay experiment that has two computing centers required Spade to be extended to allow packaged files to be relayed from one Spade to another and thus avoid the need for re-packing (which had been the initial implementation for this experiment). Figure 5 shows the deployment for two destinations (this can be generalized to multiple relays). The operation of the relay configuration is very similar to the remote warehousing with the one significant difference being that the file is already packed for the first remote Spade.

D. Fanning-out Data

Experience with the Dayabay experiment soon showed that while the relay mode works well, it was vulnerable to outages of the Beijing computer center. In order to ameliorate this, Spade was given the ability to transfer to more than one remote destinations. Figure 7 shows how this is implemented by Spade. Each transfer is independent and only when all transfers have succeeded does the local Spade delete the bundle from its cache. That way if any remote Spade has a problem and requires a re-transfer of the file it is readily available.

IV. Implementation Details

A. Application Implementation

The initial version of Spade was a standalone Java application. However the benefits of running it in a J2EE server were soon apparent and we migrated the next version to be deployed in J2EE. This simplified the database access by using EJB 3.0 and also provided a command interface by using JMX. At this point, while the various tasks within SPADE were handled in their own threads, each task had its own hand-crafted thread management. This was hard to maintain and so the next major update was to use a common thread codebase. While this update seemed to point towards moving to a workflow standard, a survey of available standards and implementations did not reveal anything that appeared to meet the needs of SPADE, therefore a custom workflow engine was created to provide the necessary common thread codebase.

During research into workflows in order to provide a generic analysis engine for the light source analysis, it became apparent that BPMN 2.0 [6] was mature enough as a workflow standard and included all the necessary features that Spade could be implemented using that. Taking this standard as the starting point, the existing workflows in SPADE were captured in this notation and then the existing custom workflow engine was modified in order to support those part of the standard

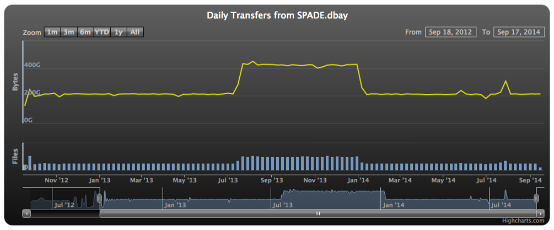

Figure 8: The transfer of files away from the Dayabay experiment over the last two years.

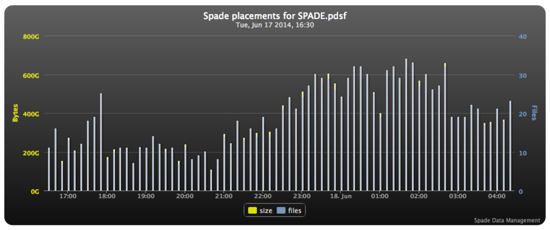

Figure 9: The placement of files in the warehouse, per 10 minutes, while Spade was running with the LCLS.

that were needed in SPADE. Figure 7 shows the result of capturing Spade's main workflow using BPMN. This is the version discussed in Section II.

Now that Spade's workflows are completely described in BPMN, the next direction of development for the application is to test whether any of the existing BPMN engines can support the complete set of requirements for the application. If such an implementation is found to exist then the obvious move it to replace the custom engine with that one.

B. *Implementing Customization*

As the number of uses of Spade has increased, the areas that are experiment or environment specific have become clearer. First and foremost is the management of metadata. While IceCube needed to support GCMD, DayaBay had not such requirement and the light sources needed a metadata that could support multiple experimental sites in one warehouse. The approach taken to fulfill this disparate requirements was to exploit the J2EE entity beans mechanism and provide a MetadataManager interface whose implementation could be selected at deployment by a suitable configuration file. Experiments where then free to either implement their own subclass or use one of the default MetadataManager subclasses provided by the project.

This approach of providing extension points by means of an interface is also used to manage such tasks as file transfers, placement policies and the mapping of a semaphore file to its data location. The Spade project itself provides at least one default implementation of each interface and, when there is a clear demand for particular implementation, it can provide these as part of the project. For example, there are implementations of the file transfer interface that use `scp`, `bbcp`, `gridFTP` and Globus Online.

V. Results and Experiences

The Spade project has been an essential part of two High Energy physics experiment that have provided significant results in the last few years [7], [8]. Both of these experiments have tested Spade's ability to handle a large separation between experiment and computer center and the network issues associated with that. Figure 8 shows that for two years Spade has been successfully moving 200GB/day of data from the experiment near Hong Kong to the computer centers at Beijing and Berkeley. Most of this time this was done by relaying the data through the Beijing computer center, but the 400GB/day section of the plot shows that Spade was equally successful when running in fan-out mode and delivering directly to both centers simultaneously.

The addition of the light source experiments as users of Spade has started to exercise different areas of its implementation. As mentioned above, there is a change of what makes up a bundle from a single file to multiple files, also the wrapping must now support a richer wrapped file than a simple `tar` archive, namely HDF5. However the most dramatic impact from these new users was the need to support high burst of data. For example experiments at the LCLS can produce 15Mb/s. Figure 9 shows how Spade handled such a detector earlier this year. It turned out that in this particular case the network infrastructure and disk access were the limiting factors.

VI. Conclusion

The success of the Spade project across an array of contexts shows that there is a need for data management software that can deliver data from dispersed facilities to one or more computer centers where a full analysis can be applied to that data. This software must be easy to use and deployable in situations that do not have a high level of computer support. The current Spade implementation has captured the essentials of this task with a core whose design has not changed over its lifetime, though its implementation has continually been updated to make use of available technologies. Given the increasing size of data produced by experiments software like Spade can only become more essential to doing science in the future.

References

[1] http://icecube.wisc.edu/
[2] http://gcmd.nasa.gov/
[3] http://gcmd.nasa.gov/KeywordSearch/Home.do?Portal=amd&MetadataType=0
[4] http://dayawane.ihep.ac.cn/twiki/bin/view/Public/
[5] http://spot.nersc.gov/
[6] http://www.omg.org/spec/BPMN/2.0/
[7] Science 342 (6161): 1242856–1242856. 2013
[8] Phys. Rev. Lett. 108 (2012) 171803