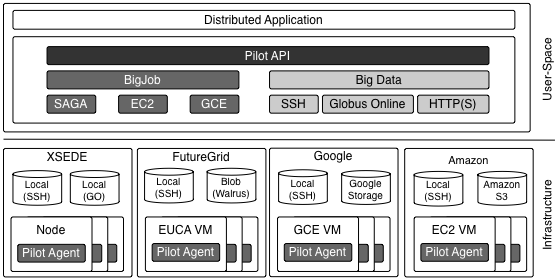
## Pilot data: Context and Motivation

Unlike libraries for HPC systems, where there was a rough (emerging) agreement on the primary architecture and programming model (MPI), there currently does not exist any consensual architecture for data-analytics at large-scale. The situation is even worse: data-analytics inherits the fragmented, heterogeneous and transient system stack characteristics of the first generation of distributed systems, e.g., data transfer and management services, grid middleware, VM hosting layers etc. Thus a layered architecture that is flexible to support intrinsic churn within different layers, yet enables insulation of upper-layers on lower layers via the right-abstractions is required; this is in contrast to the single level (i.e. library) approach that was possible for PETSc and SCALAPACK. Our abstractions are aimed at defining such an architecture for data analytics.

Pilot-Jobs have proven to be a successful abstraction for distributed and high-performance applications, whereby they decouple the work-load specification from its execution. Recent work on the P\* (Pstar) Model has provided a formal theoretical basis for Pilot-Jobs and provided a conceptual framework upon which to analyze and compare distinct implementations. Significantly, the P\* Model provided a symmetrical but logical extension to Pilot-Jobs to data, and introduced the concept of Pilot-Data, which akin to Pilot-Jobs for computational tasks, provides an efficient and extensible route to decoupling the ultimate location of data storage and/or consumption from its production. Combined, the Pilot-Job and Pilot-Data provide an effective and extensible (to notions such as affinity) abstraction for distributed dynamic data processing, i.e., data production, placement and processing. A prototype implementation of the P\* Model --- BigJob and BigData which use SAGA – an OGF community standard, is now available. The adaptor based mechanism shown in ensures that a common pilot-capability is usable on different middleware, as well as on range of cloud infrastructure. The primary reasons to use SAGA-based implementations of Pilot-abstractions as a fundamental building block are: (i) provide widely usable, integrated and extensible capabilities to a wide range of infrastructure, (ii) provides an effective runtime environment for a range of kernels, analytical routines and common patterns; common to all of these is the need for dynamic execution scenarios, e.g., for data and compute-intensive applications, (ii) transparently handles data placement and provides fundamental support for compute-data affinity.

Thus a layered architecture that is flexible to support intrinsic churn within different layers, yet enables insulation of upper-layers on lower layers via the right-abstractions is required; this is in contrast to the single level (i.e. library) approach that was possible viz., PetSC and SCaLAPACK. Furthermore, many aspects of how we deal with data processing are coupled with the model of data storage, access and movement. An initial set of capabilities that the SPIDAL via the Pilot-API will provide include: common interface for accessing data, transport data, expression of compute-data affinity and the logical placement/holding of data (which provides capabilities for dynamic data). The challenges of providing a common access layer to diverse infrastructure --- whether they be software layers (middleware), or semantically different infrastructure for data-intensive computing are: (i) determining the optimal common set of functionality, so as to provide the required range of functionality whilst not settling for the lowest common denominator, (ii) providing a “native” or “native-like” access layer, (iii) preserving performance across back-ends yet exploiting specific system capabilities, and (iv) although they present a logically unified view --- mounted as a repository to the compute clusters and thus presented as “regular” file systems, there are issues of integration of with programming models.

The last challenge requires elaboration, given the proliferation of possible infrastructure. So although the complete list of specific data infrastructure that SPIDAL will integrate with is currently not determined, the overall ecosystem and infrastructure categories is. For example, the Pilot-API must work on infrastructure that spans traditional parallel file-systems (Lustre, GPFS), SQL data-bases, distributed file systems supporting data-parallelism (Google File Systems, HDFS, Cosmos), federated file-systems (GFFS) and distributed storage systems (SRB/iRODS). The advantages of a library such as SPIDAL will ensue only if it can be used on a variety of data stores, transport and access mechanisms. The design of adaptors thus becomes a significant research and development challenge. For back-end storage systems, there exist different attributes which unfortunately are not necessarily orthogonal attributes, hence a given infrastructure can be characterized by more than one attributes (we chose the primary/dominant attribute): (a) memory key-value stores (e.g., Redis) (b) document-oriented databases which are designed for storing, retrieving and managing document-oriented data (e.g., Couch DB, Mongo DB), (c) object-stores (e.g., Amazon S3 and OpenStack Swift), (d) NoSQL constructs such as BigTable/HBase, Cassandra and Hypertable, which take column family and column store approach to organize data. Thus, rather than make specific storage choices, SPIDAL will be developed to support data-parallel or repository centric models, and research different compute placement strategies taking into account available networks and relative location of data. This research is closely related to data movement as one needs to change between storage modes between a MPI-IO parallel file system, the data-parallel file model and the OpenStack style object store. The integration of these adaptors with the overall software stack and the production environment becomes a non-trivial engineering task.