Abstractions for Adaptive Biomolecular Simulation Algorithms

***Abstract*—There is a fundamental need to expand the reach of biomolecular simulations in order to understand many unsolved but challenging problems. In order to bridge this gap, biomolecular simulations must grow to support larger systems, longer time scales as well as to engineer better fidelity in the modeling of complex phenomena. With the end of traditional Moore's Law scaling, running simulations of longer timescale physical processes cannot continue to rely on simple hardware enhancements. Coupled with the desire to simulate complex phenomena with adequate fidelity, there is a need to support a wider range of simulation capabilities than is currently easily feasible. In particular, the ability to compose high-level simulation algorithms incorporating high-level parallelism, loosely coupled ensembles of independent simulations, and leveraging the statistical nature of molecular processes can provide a means to address these challenges of scaling. In this paper, we (i) analyze several common methods/algorithms used in biomolecular simulations, (ii) implement adaptive workflows for three production use cases from current biomolecular simulation research, (iii) analyze the performance and functionality of our toolkit implementation for these workflows, and (iv) identify a lightweight set of functions necessary and sufficient to express these adaptive algorithms in a simple yet powerful manner.**

I. Introduction [SJ, PK]

[What:] Problem statement. Current computational methods for solving scientific problems in biomolecular simulation are at or near their scaling limits. This manifests in three areas: strong scaling limits, the ability of platforms to coordinate large numbers of jobs, and the current level of user interaction required. Overcoming these barriers will require advances in the scale and sophistication of simulations. For example, scaling workflows to 100x - 1000x current levels will require both execution platforms that can support much greater degrees of parallelism, but also more automated adaptivity to encode logic that is currently performed by users into the workflow itself. Such adaptivity is a critical requirement for scaling in many scientific applications: the pattern of execution cannot be simply enumerated a priori but depends upon intermediate results and control logic embedded in the workflow. To achieve scalability, efficiency and reproducibility, such adaptivity cannot simply be performed via user intervention. This paper aims to define a simple high-level formalism to capture adaptive workflows and analyze performance considerations associated with adaptivity to enable scalable adaptive workflows..

[How:] To guide the design and implementation of capabilities to execute high-level biomolecular simulation algorithms in a scalable and adaptive manner, we (i) identify a set of adaptive workflows used in production by biomolecular simulation groups, (ii) provide a toolkit implementation to enable succinct and extensible expression of these workflows, and (iii) analyze the coordination/communication/execution requirements associated with each use case. We provide a common runtime system which provides dynamic resource capabilities and measure the scalability of these implementations using the common runtime system in a set of experiments to guide further design of scalable and adaptive workflows. We identify a minimal set of functions to express most common types of adaptivity utilized in biomolecular simulation in a simple yet powerful manner.

Hypothesized Outcomes (experiments should help support/negate)

- Proof-of-concept/existence: Why this is needed and why can't existing workflow systems address this.
- Functionality
- Performance
  - (i) cost of generality. (ii) price of shallow integration, (iii) real time.
  - scalability usual strong and weak scaling.
  - price of adaptivity?
- Usability
- Systems and Software Design Issues
  - Integration hell vs new software
  - Deep versus shallow integration between layers for performance optimization
  - Compare w/ TCL integration in NAMD

II. Scientific Motivation

(problem space)

Discussion of some high-level "simulation algorithms" already in broad use and the associated adaptivity. A number of molecular simulation workflows require the management of large ensembles of simulation workflows, and many powerful new methods adaptive require that these workflows be adaptive.

- Replica exchange (or parallel tempering) is one of the most common methods to improve scaling of large scale molecular simulations. In replica exchange, a series of simulations running under different simulation conditions (such as different temperatures or pressures) are loosely coupled together, exchanging configurations stochastically in a way that preserves the correct probability distribution for each simulation condition. Since in many cases configurational sampling is faster at some of these simulation conditions (for example, kinetics are typically faster at high temperature. However, as new replica-exchange based algorithms are developed that sample over three or four different simulation conditions, or even over multiple molecular models, the number of simulations in the ensemble increases to the power of number of variables. In order to manage this size and complexity, adaptive choices must be made about how to

switch between ensemble members and which of all the possible simulations should be run at any given time.

- Metadynamics and expanded ensemble dynamics are another class of algorithm, where individual simulations jump between simulation conditions. Additional weights are needed to force the simulations into the desired probability distributions in state space, usually even sampling in all the simulation conditions, and such weights are learned adaptively using a variety of schemes. However, because the movement between state spaces is essentially diffusive, if multiple state spaces are used, sampling is slowed down. This can be handled using "multiple walker" approaches, with more than one simulation exploring the same state space. These otherwise independent simulations communicate by collectively building up the weights that result in even sampling of the simulation conditions of interest.
- Markov state modeling is another important class of simulation algorithms, this time for determining kinetics of molecular models. Using an assumption of separation of time scales of molecular motion, the rates of first-order kinetic processes are learned adaptively. In a typical Markov State Model simulation, a large ensemble of simulations, typically tens or hundreds of thousands, are run from different starting points, and similar configurations are clustered as states. The rates of transitions between these states are estimated by observing which. The entire kinetic behavior can then be inferred even though individual simulations perform no more than one state transition. However, the choice of where new simulations are initiated to best refine the definition of the states, improve the statistics of the rate constants, and how to discover new simulation states requires a range of analyses of previous methods, making the entire algorithm highly adaptive.
- We should look into other adaptive execution based applications, uncertainty quantification and other domains (i.e., outside of biomolecular simulations).

III. Computational Motivation

A. Why was there a need/proliferation of monolithic workflow management systems?

a) Workflows were for Big Projects:: Historically workflows were required by, if not the preserve of large projects, that had complex infrastructure requirements which needed support both managing rudimentary infrastructure capabilities and adapting applications to the rudimentary infrastructural capabilities. This led to the birth of Pegasus (based upon Condor) in 2001 and VDToolkit (predecessor to Swift) around that time.

b) Missing abstractions:: When workflow systems were originally conceived, there were no well established abstractions and resource management concepts such as Pilot-Jobs or even standardized access-layers. Thus, the first generation of workflow systems needed to be end-to-end integrated systems supporting everything from application development/interfaces dynamic resource management and access to heterogeneous middleware layers. Specific example of dynamic resource management for workflows: At the turn of the century, there was a proliferation of workflow systems. Each workflow system needed better and finer grained support for advanced resource management. The rise of workflow systems is orrelated with the proliferation of Pilot-Jobs/LHC computing. [Similarly, in the past few years, there seems to be an upswing in the number of workflow systems that are oriented towards HPC systems, e.g. Fireworks]

c) Managing Heterogeneity:: Limited ability to manage heterogeneity at multiple levels. Services and interfaces.

A partial analysis of DAGMan [escience2009] and attempts to separate the resource access layer from DAGMan functionality shows that the absence of well defined interface had important consequence.

d) Programming languages and programming models: The absence of general purpose scientific languages such as Python motivated the need for specific / customized workflow languages and possibly specific workflow programming models.

There were a few important workflows that were worth the significant efforts in integrating with workflow systems. For example of 60 workflows that LIGO needs/has, exactly one is workflow system enabled. In general the number of workflows that need supporting have increased, but the effort to adapt existing workflow solutions has remained high.

B. But why the building block approach to workflows now?

e) Need for workflows are much more pervasive and no longer confined to big projects:: This is indicative of the fact that many science applications often representing individual PI science teams, require non-traditional usage of HPC/exascale systems and demand unprecedented scales. Big Science can come from small teams. There are technology factors (such as end of moores law, Dennard scaling, and limited strong/weak scaling), and application factors (ensembles, statistical uncertainty) that point to the need for multiple applications.

f) Advances in the science of cyberinfrastructure:: In the past 10 years, there have been significant advances in the science of pilot jobs, interfaces etc. For example, Ref [pilot-jobs] and Ref. [software-x]....

g) Last mile customization often has significant upstream consequences:: The point of integration between capabilities is not always uniform. e.g., supporting adaptive execution of ensembles. Although adaptive execution of ensemble workflows may seem like an just another functional requirement resources this has profound consequences.

h) Proliferation of Domain/Specific Workflow Systems:: There are good reasons why there has been a proliferation of smaller domain specific workflow systems (e.g., Seisflow (Tromp), ExTASY (Clementi), RepEx (Trekalis) .....) that arent as resourceful as the workflow systems developed by the Big Projects. The proliferation of many bespoke and domain specific workflow systems cannot be stopped, nor should they be: where they make sense, they should be facilitated and managed, as opposed to hindered. This requires providing building blocks so they can focus on X, as opposed to all teams repeating and developing the same features poorly.These building

blocks are needed by tool developers and application/domain scientists alike.

i) Engineering for Performance:: Performance is a multi-dimensional parameter; thus there are many direction-s/dimensions in which it needs to be tuned. Cant tune entire stack. With a modular building block approach there is no need to actually. Building Block approach provides modularity. In a workflow management system not all layers of the stack equally contribute to performance; closer to the system level the more critical from a performance point-of-view.

This is not to say that we shouldnt have end-to-end workflow systems. They retain a role. However, there are two classes of users that are not being served with the current set of monolithic workflow tools and systems. (i) Workflow system and tool developers, and (ii) easy development and extensibility to a large number of workflows.. Many of which grow or evolve as the science advances (thus reiterating the importance of end-user development). The effort to integrate the remaining 59 workflows of LIGO is arguably uneconomical.

The vision is that as a consequence of well-defined, designed and implemented building blocks, developing workflows will be closer to pythonic programming using scientific libraries. Ideally each building blocks can be provided as library, where these libraries have well defined interfaces and semantics but each library is engineered for performance on platforms. This will prevent workflow vendor lock-in and rigidity in programming models. RADICAL-Pilot [RP-Cray] demonstrated how building blocks can be developed to have semantically well defined functionality [RP-arxiv] yet be tuned for performance.

C. Challenges of the building blocks approach?

- Performance versus generality: Can adequate performance be achieved whilst aiming for components that are general and extensible to be used by multiple distinct applications and tools.
- Integration complexity: A multi-component design approach has the challenge of
- Impedance mismatch: Similarly there is a risk of functional gaps..

IV. Related Work

(solution space)

Many tools have been developed to run applications on HPC systems. Most of these tools [] are configured to execute predefined applications on static resources.

- Dakota as a nice example of high-level algorithm implementation (simulation algorithms above the level of individual simulation runs) but not a full stack for adaptive ensemble simulation (e.g. not the same kind of runtime capabilities).
  - Dakota: https://dakota.sandia.gov/release-notes-headings/uncertainty-quantification-uq
  - Also from Sandia: http://www.sandia.gov/uqtoolkit/
- Landscape of tools that aim to support adaptive workflows [All]

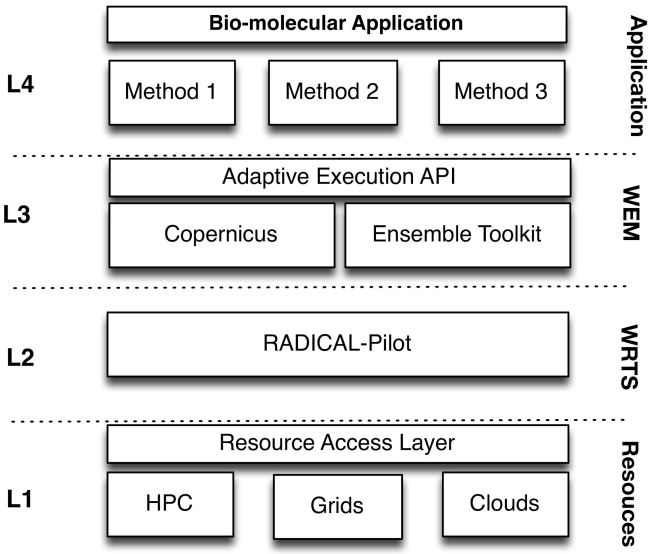

Fig. 1. Application stack

V. Understanding Adaptivity [PK, VB/SJ]

***SJ:some stuff from google docx, viz., Types of Adaptivity: can move here

Here we define "adaptive" execution workflows as ones where the tasks executed depend on the intermediate results and thus cannot be fully enumerated in advance. We briefly categorize adaptivity as 1) fully specified sequence of execution but resource utilization determined at runtime (an example of "late binding"), 2) fully specified sequence of execution but the number of times a given statement is executed is determined at runtime, 3) the high-level logic flow is determined at runtime (e.g. if/then/else controlling the tasks to be executed). ***SJ:Make consistent with google doc, "types of adaptivity".

We have discussed above several examples of high-level adaptive algorithms in biomolecular simulations; theoretical work on sampling and other strategies suggests the possibility for many more, but implementation has been limited by the lack of a simple yet powerful means to express adaptive logic. In current usage, adaptivity is typically provided by user intervention and manual decision-making, but this is clearly not scalable. A requirement for scalable adaptive scenarios is thus that (i) adaptivity be encoded in the computational logic of the workflow in a fashion that is also modifiable and extensible by end users, and (ii) runtime support be provided.

Adaptivity permits greater efficiency, greater sophistication of computational strategies, and reproducibility of complex strategies. Mechanisms to capture adaptivity will enable simple and succinct expression of these types and assist in decoupling high-level biomedical simulation algorithm from both programming models and runtime systems.

VI. Computational Challenges [PK, VB/SJ]

- HPC systems traditionally designed for one monolithic application and static execution. We break both monolithic model and static execution

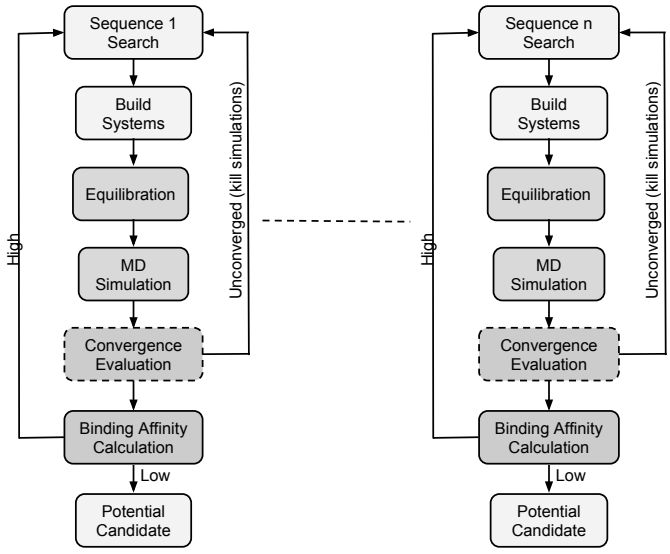

Fig. 2. Ebola pattern

- Express Adaptivity. No consistent way or form to do so.
- Need runtime system to do so. Approaches and tradeoffs (scale, flexibility etc).

***SJ:What additional challenges does adaptivity present (beyond regular dynamic execution of ensembles)?

This section will likely move before Use Cases.

VII. Production Use Cases

Here we have chosen several examples of adaptive workflows currently used in production by different biomolecular simulation groups. We provide toolkit functionality to enable more succinct expression of these workflows within the Ensemble Toolkit framework and analyze their functionality and performance.

A. Use Case 1

Use case 1: computational design of peptide inhibitors of Ebola virus entry ("Ebola"). The Ebola workflow can be mapped to an iterative 6-stage pipeline with multiple decision points. Each stage of the pipeline may have different number of ensemble of tasks.

- **Stage 1** consists of one task (i.e. ensemble size=1) which performs a sequence search using the Rosetta software to generate multiple simulation candidates.
- **Stage 2** consists of one task which performs transformations of Rosetta output to Amber compatible data.
- **Stage 3** consists of multiple tasks (i.e. ensemble size¿1) that execute equilibration scripts.
- **Stage 4** consists of multiple tasks that perform production MD runs using Amber.
- **Stage 5** consists of a single task which analyses that output of the MD runs to separate "good", "bad" candidates. Stage_5 consists of a branch which operates on the output of stage_5 to decide if the process needs to be restarted (unstable) or can proceed (stable).
- **Stage 6** consists of multiple tasks (?) that perform MMPBSA analysis to generate data that is used to decide which candidates can be kept and which to be used for the next iteration. Stage_6 also consists of a branching function which performs the above decision.

The adaptivity in this use case comes from the ability to decide the next stage *(i)* based on the output of the previous stage *(i-1)* (or combination of outputs from previous stages).

It is not exactly sure where/when the iteration stops from the diagram in the slides

```
# Each pipeline of the Ebola workflow has the following
# structure
pipe(siminput):

  while (True):

    # Search for the sequence of interest
    a = reduce(seq_search, siminput)

    # Build system for the sequence
    b = reduce(build_sys, a.output)

    # Equilibriate
    c = reduce(equil, b.output)

    # Run MD simulation on the trajectory input
    d = async reduce(mdkernel, c.output)

    # Check the simulation output for convergence
    while not reduce(any_true, e = reduce(check_converge, d
     .output).output):

      # if not ready yet, wait for another increment worth
     of trajectory output data
      reduce(waitforincrement, e.output)

    # Check if convergence is reached
    if e.output > conv_threshold:

      # Restart pipeline
      continue

    # Calculate BAC
    f = reduce(bac, e.output)

    # Check if BAC output is greater than expected
     threshold
    if f.output > bac_threshold:
      continue
    else:
      # Success
      break

z = map(pipe,siminputs)
```

Listing 1. Pseudocode for individual pipeline of the Ebola workflow

B. Use case 2:

```
# Each pipeline of the EE workflow has the following
     structure

pipe(siminput):

  while (True):

    # Run MD simulations on the input trajectory
    a = reduce(mdkernel, siminput)

    # Generate weights and exchange
    b = reduce(gen_weights, a.output)

    # Exit condition ?


z = map(pipe, siminputs)
```

Listing 2. Pseudocode for Expanded Ensemble workflow

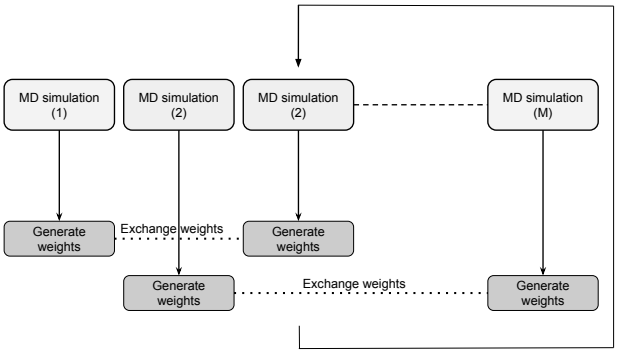

Fig. 3. Expanded Ensemble pattern

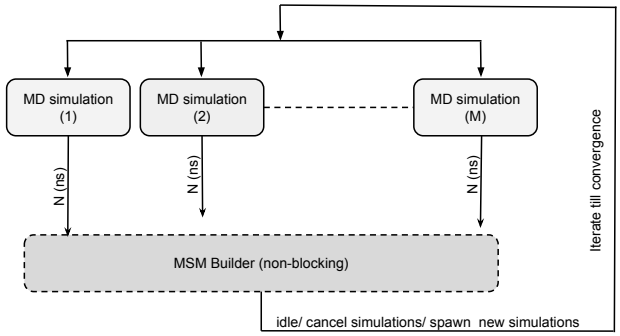

Fig. 4. Adaptive MSM pattern

Worfklow consists of multiple pipelines executing concurrently. Each pipeline consists of 2 stages: simulation and exchange.

- **Stage 1** consists of MD simulations that produce trajectory data.
- **Stage 2** consists of calculation of weight to be assigned to each simulation and exchange of this information with other pipelines of the workflow.

fill in with Michael's use case

C. Use case 3:

adaptive sampling for efficient Markov State Model construction to describe the folding of the Fip35 WW domain. Markov State Models provide an attractive way to encode dynamic processes such as protein folding into a set of metastable states and transitions between them. In computing these models from simulation trajectories, but the metastable state definitions and the transition probabilities have to be inferred. It has previously been shown both theoretically [**?**], [**?**] and prospectively [**?**] that "adaptive sampling" can lead to more efficient MSM construction as follows: provisional models are constructed using intermediate simulation results, and these models are used to direct the placement of further simulation trajectories. Here, we have applied adaptive sampling prospectively to the folding of the Fip35 WW domain as a moderately challenging but tractable proof of concept. This workflow consists of an iterative pipeline in two stages: ensemble simulation and MSM construction to determine optimal placement of further simulations. The pipeline is iterated until convergence of the resulting Markov State Model.

- **Stage 1** consists of multiple (i.e. ensemble size ¿ 1) MD simulations that produce trajectory data. These simulations will trigger analysis every N_total ns of aggregate simulation time, although individual simulation lengths should each exceed some decorrelation time tau.
- **Stage 2** consists of a single task that uses the MSMBuilder tool to cluster the MD simulation output trajectories and create an MSM. This stage also generates new simulation start states to be run in the next iteration for more efficient convergence of the MSM than continuing the original trajectories.

```
# run one round of simulation and clustering first (both
    decorrelation and making our isnotconverged evaluate
    properly)
a = map(mdkernel, siminputs)
b = reduce(MSMsample, a.outputs)
trajinputs = b.outputs

while reduce(isnotconverged, b.outputs):
  c = async map(mdkernel, trajinputs)
  # Run MSMsample at intervals; outputs is new trajectory
    inputs, while outputs_stop is which trajectories to
    stop
  # when MSMsample says to stop some trajectories, we're
    ready for a new round of the outer while loop

  while not reduce(any_true, (d=reduce(MSMsample, c.outputs
    )).outputs_stop):

    # if not ready yet, wait for another increment worth of
      trajectory output data
    reduce(waitforincrement, c.outputs)

  cancel(c, d.outputs_stop)
  trajinputs = d.outputs
```

Listing 3. Pseudocode for adaptive MSM pattern

VIII. Experiments [VB/SJ, PK]

A. Experiment Setup

- Resources to be used [Bluewaters, Stampede, Comet]
- Physical system to be used – why ?

***VB: the following experiments subsections can be grouped later

B. Ebola experiments

C. Extended Ensemble experiments

D. MSM experiments

Performance measurements can be made along the following parameters:

- Wallclock time to convergence
- Acceleration factor - Ratio of length of shaw trajectory used to calculate convergence and aggregate simulation time used from MSM method
- strong, weak scaling experiments ***VB: this experiment might depend on the whether the optimal number of simulations to be executed is system-independent/dependent

***VB: General notes:

- Metric 1: expressivity/usability (how encapsulate this?)
- Metric 2: Scalability and performance
  - Linearity in scaling ?

– At what scale does this linearity break ?
– Does adaptivity introduce latencies? (more of an L2 concern)

IX. Proposed operations

Based on these use cases, we have developed the following set of functions that are necessary and sufficient to express the underlying adaptivity in our production examples. We believe these functions are more general as well and should be able to capture adaptivity for a broad class of adaptive ensemble problems. All the functions operate on compute kernels, denoted f, g, and h, and take inputs, possibly arrayed, denoted **inp**. The **map** and **reduce** operations provide basic parallel functionality similar to MapReduce but with the important difference that reduce() returns a variable-dimension output. They also enable the adaptive capability provided by **while**() and **if**(). The **async**() and **cancel**() operations add capabilities for asynchronous tasks that can operate on intermediate outputs, for instance analysis processes that monitor simulation outputs and return decisions on whether to cancel them and spawn new simulations.

- **map**(f, **inp**): Run kernel f in parallel on each of the input array members in **inp**. Produces an output of equal dimension to **inp**.
- **reduce**(f, **inp**): Run kernel f on all of the input array members in **inp**. This produces an output of variable dimension. This function is used for operations such as clustering.
- **while**([reduce] f, **inp**): **run**(g, **inp2**). Evaluate f on **inp**. While f is true, run g. If **inp** is parallel and f has an equal output dimensions, different elements of the while loop may run for different numbers of iterations. If f has a single output but **inp2** is parallel, all elements of the while loop are run for the same number of iterations.
- **if**([reduce] f, **inp**): g, **inp2**; else h, **inp3**. Evalute f on **inp**, if true run g and if not run h. If **inp** is parallel, the same considerations apply as to **while()**.
- **async** [operation]: specifies that the operation runs in the background and control returns to the next block in a control-flow formalism; in a data-flow formalism, the operation can write intermediate as well as final outputs.
- **cancel**(operation handle, boolean mask): takes a mask of dimension equal to the parallelism of the operation handle and cancels the subtasks at locations where the mask value is True.

***SJ:Propose to call the derived API, "Adaptive Ensemble API"

X. Discussion, Conclusion and Future Work

Software and Systems: Discussion of effort to take (an) existing Workflow system and modify to meet requirements (there is more than one way to do it), versus integrating an existing workflow system with capabilities developed here.

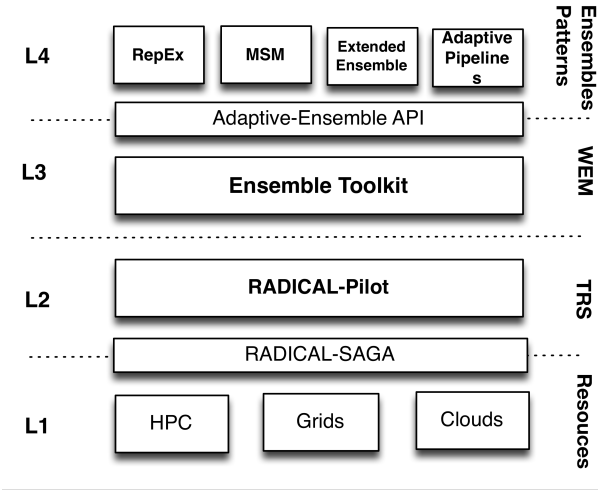

Fig. 5. An overview of the design and components used in this paper ***SJ:For comparison with Fig.1 and discussion