

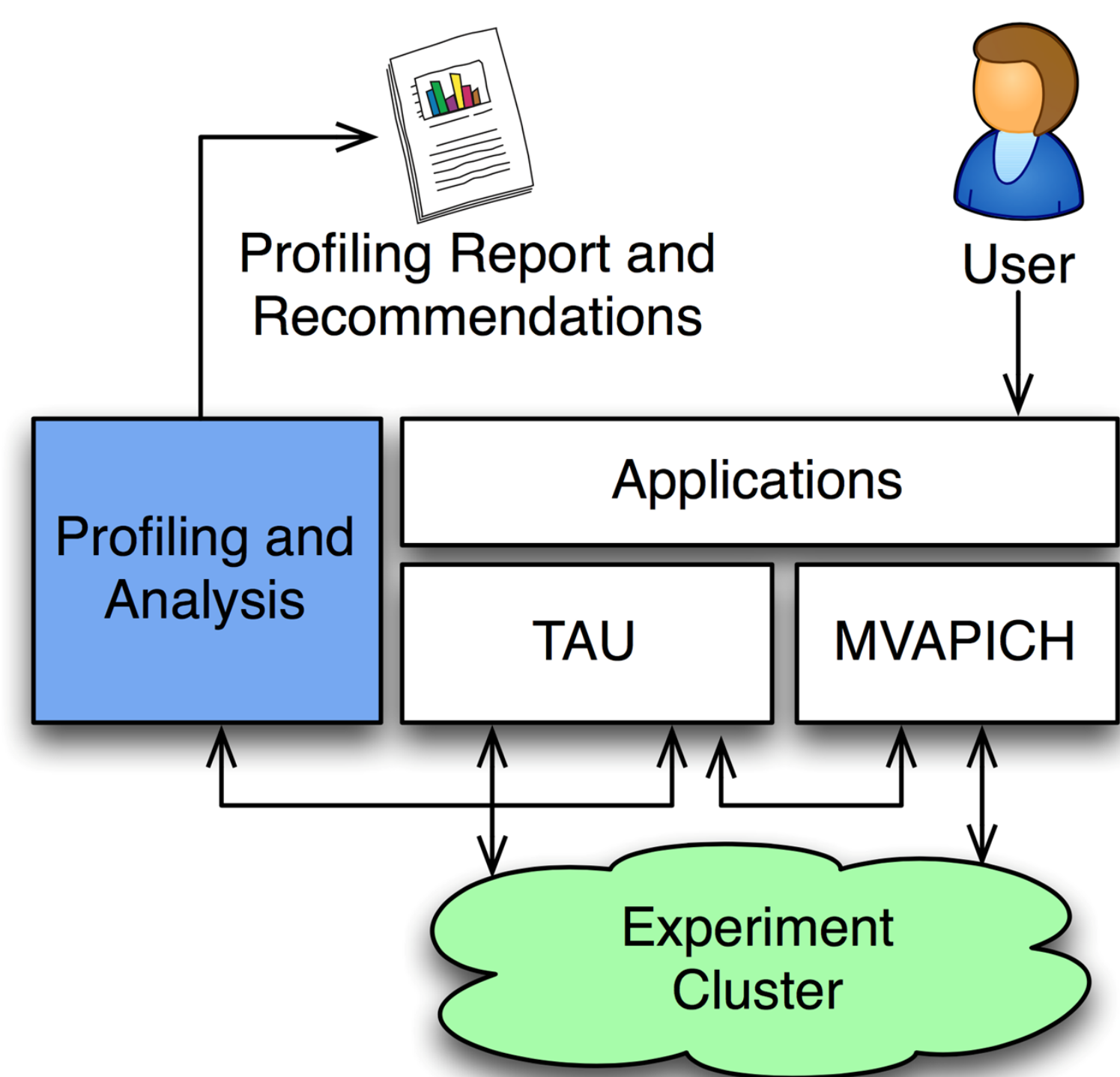
# SI2-SSI (2018): Collaborative Research: A Software Infrastructure for MPI Performance Engineering: Integrating MVAPICH and TAU via the MPI Tools Interface

H. Subramoni, P. Kousha, A. Ruhela, S. Chakraborty, and D.K. Panda  
The Ohio State University  
<http://mvapich.cse.ohio-state.edu>

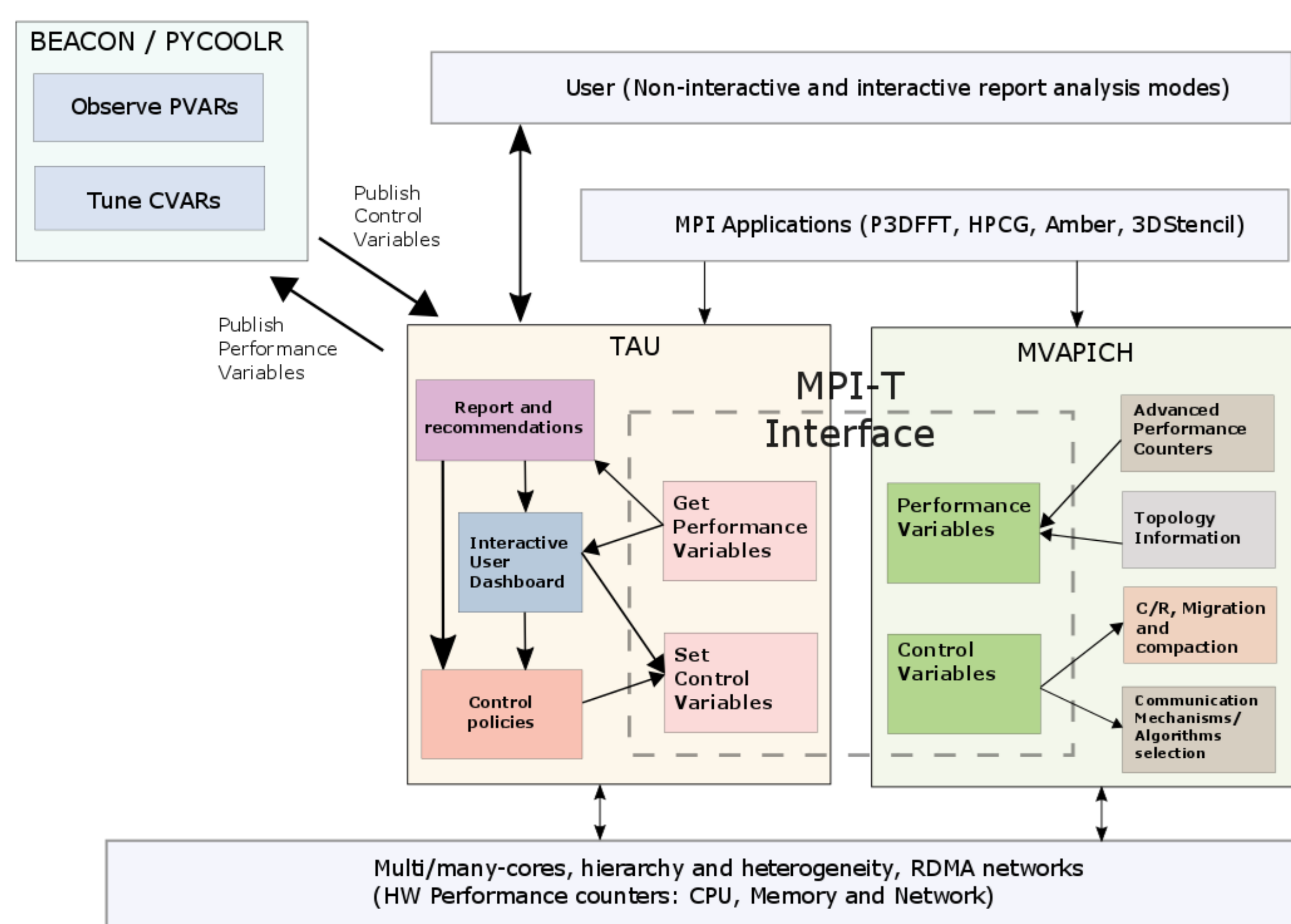
S. Shende, A. D. Malony, A. Maheo, and S. Ramesh  
University of Oregon  
<http://tau.uoregon.edu>

## Research Challenges

**Creating an MPI programming infrastructure that can integrate performance analysis capabilities more directly, through the MPI Tools Information Interface, monitor Performance metrics during run time, and deliver greater optimization opportunities for scientific applications.**



## Proposed Approach



- Recommendation module and control policies are realized using a generic plugin architecture inside TAU
- Plugin design allows custom policy modules to be implemented and loaded as needed

## Policy Engine for MPI\_T Plugins

### Motivation for a policy engine

- The core tuning / recommendation logic for most plugins would likely consist of simple condition checks using PVARs and setting of CVARs
- We would like to take advantage of this common structure by:
  - Defining a policy engine (generic autotuning plugin) with PVARs and CVARs templated out
  - Providing a simple JSON-based rule script that plugin writers use to fill in the template parameters for PVARs, CVARs, and custom thresholds used in tuning

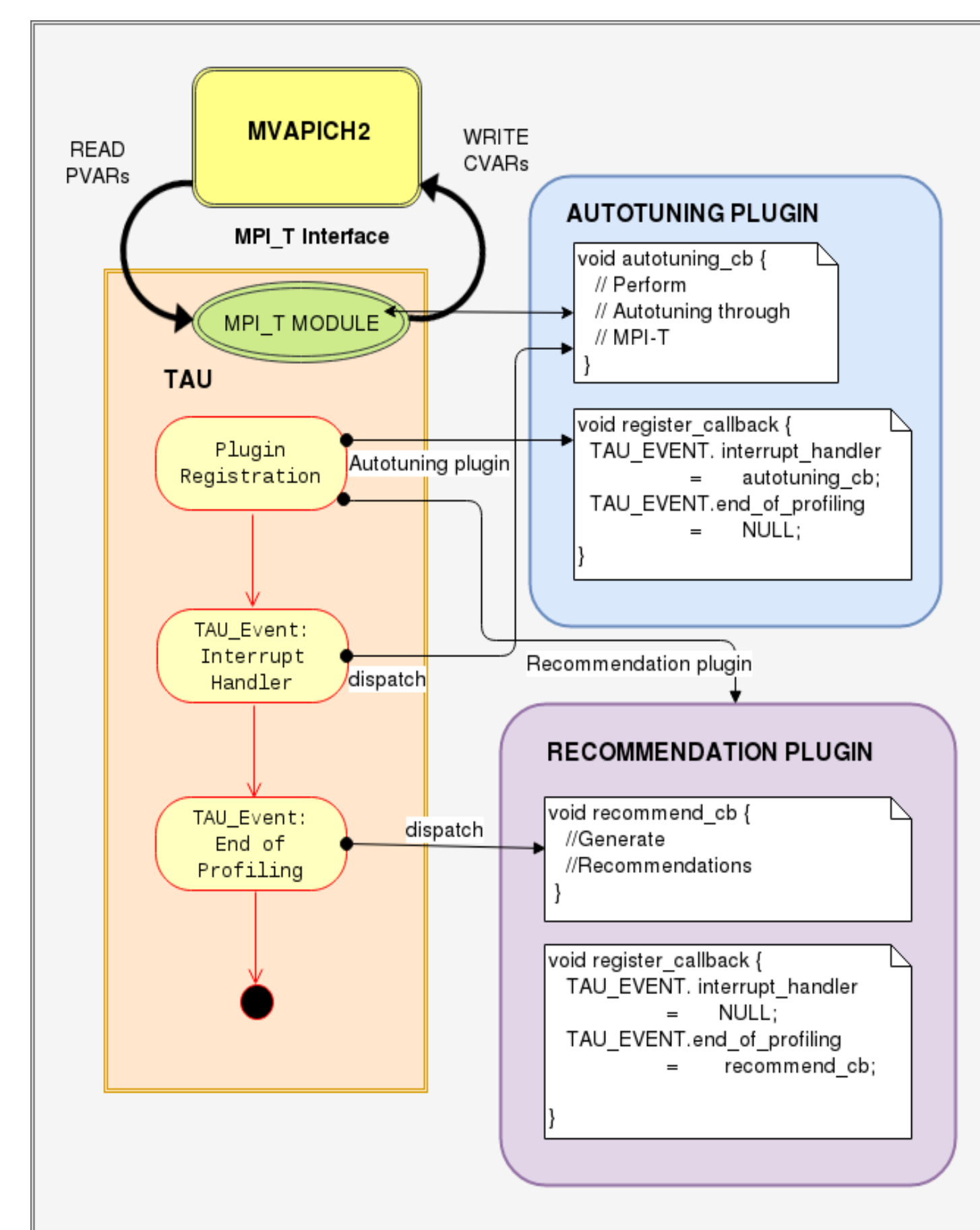
### Rule design for the policy engine

```
rule {
  num_pvars: ...
  operation: {
    condition: {
      stmt: if,
      leftoperand:pvar1,
      rightoperand:value,
      operator:==,<,>,...
    },
    result: {
      leftoperand:cvar1
      rightoperand:value
      operator:=
    },
    else: {
      leftoperand:cvar1
      rightoperand:value
      operator:=
    }
  }
}
```

- We use JSON for describing the rule itself:
  - Input: PVARs to be tested for a specific condition
  - Output: CVARs
- In essence, the rule is describing a recursive “if-condition” check with user supplying the operands and operations
- The policy engine reads the rule and computes the actual autotuning / recommendation logic that the rule represents
- Benefits of a policy engine:
  - Code redundancy is avoided
  - Users need not write

## TAU Plugin Infrastructure Design

- Plugins are written in C/C++ and implemented as shared library modules
- Plugins register callbacks to salient user-defined events in TAU
- Using the environment variables TAU\_PLUGINS and TAU\_PLUGINS\_PATH, multiple plugins can be loaded in order



- Plugin events currently supported are:
  - FUNCTION REGISTRATION
  - ATOMIC EVENT REGISTRATION
  - ATOMIC EVENT TRIGGER
  - INTERRUPT TRIGGER
  - END OF EXECUTION
- Plugins can register callbacks for more than one event

## Autotuning Plugin for SNAP

### TAU Autotuning plugin design for freeing unused VBUFs

- Plugin registers a callback for the INTERRUPT TRIGGER event that is triggered when TAU has been configured to collect PVARs at regular intervals by setting TAU\_TRACK\_MPI\_T\_PVARs
  - Plugin has access to PVARs that represent the quantity of unused MVAPICH2 Virtual Buffer (VBUF) resources
  - Plugin sets MVAPICH2 CVARs to enable pool control when it detects that the quantity of unused VBUFs has breached a user-defined threshold

### Large scale study with SNAP particle transport application that benefits from a higher Eager threshold and freeing of unused VBUFs

- Increasing the Eager threshold improves the point-to-point performance at the cost of increased VBUF memory usage inside MVAPICH2
- VBUF memory usage increases as larger VBUFs may need to be allocated to store Eager messages in transit

Run	Number of Processes	Eager Threshold (Bytes)	Runtime (secs)	Total VBUF Memory (Bytes)
Default	1024	MVAPICH2 Default	47.3	3,322,067
Eager	1024	20,000	42.2	3,787,050
TAU autotuning plugin	1024	20,000	42.9	2,063,421

## Enhanced MPI\_T Support in MVAPICH2 and Utilizing it in TAU

- MVAPICH2 has multiple optimized designs for collective operations.
- Choosing the algorithm that deliver the best performance for a given application is complicated and depends on several factors like message size, size of the job, availability of advanced hardware features etc.
- PVARs to measure various aspects of collective algorithm usage like bytes/messages sent/received have been added for all available algorithms for:
  - Alltoall, Alltoallv, Gather, Gatherv, Scatter, Scatterv, Allgather, Allgatherv, Broadcast, Reduce, Allreduce, Reduce\_Scatter, and Barrier

Name	NumSamples	MaxValue	MinValue	MeanValue	Std. Dev.
Message size for allreduce	2,401	8	8	8	0
Message size for reduce	1	8	8	8	0
mem_allocated (Current level of allocated memory within the MPI library)	6	7,228,797	7,228,797	7,228,797	0
mem_allocated (Maximum level of memory ever allocated within the MPI library)	6	7,228,797	7,228,797	7,228,797	0
mpit_progress_poll (CH3 RDMA progress engine polling count)	6	78,157,187	14,156,144	51,872,114	22,795,852.854
mv2_coll_algather_bytes_recv (Number of bytes recv by default algorithm of algather collective)	6	68	68	68	0
mv2_coll_algather_bytes_send (Number of bytes send by default algorithm of algather collective)	6	28	28	28	0
mv2_coll_algather_count_recv (Count of messages recv by default algorithm of algather collective)	6	3	3	3	0
mv2_coll_algather_count_send (Count of messages send by default algorithm of algather collective)	6	3	3	3	0
mv2_coll_algather_rd_bytes_recv (Number of bytes recv by rd algorithm of algather collective)	6	68	68	68	0
mv2_coll_algather_rd_bytes_send (Number of bytes send by rd algorithm of algather collective)	6	28	28	28	0
mv2_coll_algather_rd_count_recv (Count of messages recv by rd algorithm of algather collective)	6	3	3	3	0
mv2_coll_algather_rd_count_send (Count of messages send by rd algorithm of algather collective)	6	3	3	3	0
mv2_coll_alldreduce_2d (Number of times MV2 twolevel alldreduce algorithm was invoked)	6	2,386	414	1,505,667	713.488
mv2_coll_alldreduce_bytes_recv (Number of bytes recv by alldreduce collective)	6	1,944	1,944	1,944	0
mv2_coll_alldreduce_bytes_send (Number of bytes send by alldreduce collective)	6	1,944	1,944	1,944	0
mv2_coll_alldreduce_count_recv (Number of messages recv by alldreduce collective)	6	57	57	57	0
mv2_coll_alldreduce_count_send (Count of messages send by alldreduce collective)	6	57	57	57	0

- Introduced support for new MPI\_T based CVARs to MVAPICH2
  - MPIR\_CVAR\_USE\_BLOCKING - Enables mvapich2 to use interrupt driven mode of communication progress
  - MPIR\_CVAR\_USE\_SHARED\_MEM - Enables the use of shared memory for intra-node communication
  - MPIR\_CVAR\_STRIPING\_THRESHOLD - Specifies the message size above which MVAPICH2 begins to stripe the message across multiple rails (if present)
  - MPIR\_CVAR\_USE\_COALESCE - Enables coalescing multiple small messages into a single message to increase throughput
  - MPIR\_CVAR\_RNDV\_PROTOCOL - Specifies rendezvous protocol to be used by MVAPICH2

TAU Profiling Manager (in progress)		
File Options Help		
<div><div><div>Applications</div><div>Standard Applications</div><div>Default App</div><div>Default Exp</div><div>LEASH/SHASH</div><div>TIME</div></div></div>	TrailField	Value
	CWD	/usr/local/bin/mvapich2.0
	Cache Size	25344 KB
	Command Line	./mvapich2.0 -s 35
	Ending Timestamp	152438274852712
	Executable	/usr/local/bin/mvapich2.0
	File Type Index	1
	File Name	tau
	Host Name	pegasus
	Local Time	2018-04-21T15:11:31-07:00
	MPI Processor Name	pegasus
	MPI_T CVAR: MPIR_CVAR_ABORT_ON_LEAKED_HANDLES	If true, MPI will call MPI_Abort at MPI_Finalize if any MPI object handles have been leaked. For example, if MPI_Comm ...
	MPI_T CVAR: MPIR_CVAR_ALLGATHER_PIPELINE_MSG_SIZE	The smallest message size that will be used for the pipelined, large-message, ring algorithm in the MPI_Allgather imple...
	MPI_T CVAR: MPIR_CVAR_ALLGATHER_COLLECTIVE_ALGORITHM	This CVAR selects proper collective algorithm for allgather operation.
	MPI_T CVAR: MPIR_CVAR_ALLGATHER_LONG_MSG_SIZE	For MPI_Allgather and MPI_Allgatherv, the long message algorithm will be used if the send buffer size is >= this value (...)
	MPI_T CVAR: MPIR_CVAR_ALLGATHER_SHORT_MSG_SIZE	For MPI_Allgather and MPI_Allgatherv, the short message algorithm will be used if the send buffer size is < this value (...)

## Recommendation Plugin for MiniAMR

- Plugin infrastructure can be utilized to generate performance recommendations as metadata on ParaProf
- Recommendation plugin registers a callback for the END OF PROFILING event that is triggered when TAU has finished profiling
  - Plugin has access to all the profiling data collected

### Recommending usage of SHaRP hardware offloading of MPI\_Allreduce for MiniAMR

- MVAPICH2 supports hardware offloading of MPI\_Allreduce using Mellanox' SHaRP protocol (disabled by default)
- MiniAMR uses MPI\_Allreduce for small message size (8 Bytes)
  - Prime candidate to benefit from hardware offloading of collectives
- Recommendation plugin combines profiling and message size information gathered by TAU from the PMPI interface to recommend the user to enable MPIR\_CVAR\_ENABLE\_SHARP

Run	Number of Processes	Runtime (secs)
Default	224	648
SHaRP enabled	224	618

## Future Work & Research Dissemination

- Further enhancing the MPI\_T support in MVAPICH2 and co-designing TAU to take advantage of it
  - We are exploring the potential of the MPI Tools Information Interface to support scenarios that require extremely fine-grained tuning, such as:
    - Tuning low-level network protocols at a fine granularity
- Continue to study the benefits of utilizing CVARs exposed by MVAPICH2 at application level on large supercomputing systems
- Study challenges in providing an interactive performance engineering functionality for end users
- Release MVAPICH2 and TAU with enhanced support

This work was supported by the NSF under the ACI-1450440 & ACI-1450471 grants. This work used the Extreme Science and Discovery Environment (XSEDE) which is supported by National Science Foundation grant number ACI-1053575. This work used allocation grants TG-ASC090010 & TG-NCR130002.

