OPENFABRICS ALLIANCE

2023 OFA Virtual Workshop

/CHECK: LEVERAGING RDMA AND MALLEABILITY FOR APPLICATION-LEVEL CHECKPOINTING IN HPC SYSTEMS

Jophin John

Chair of Computer Architecture and Parallel Systems (CAPS)
Technical University of Munich, Germany

OVERVIEW

- Dynamism in Checkpointing
- *i*Check - ***Invasive*** Checkpointing – Intro to Invasive Computing
- Invasive Infrastructure
- Dynamism in *i*Check
- Results
- Into the Future

A CASE FOR *DYNAMISM* IN CHECKPOINTING

- Plenty of techniques for fault tolerance
- Focus on Application-level checkpoint restart (widely used in Simulations)
- Systems and Applications are becoming malleable in HPC
- *Dynamism* in checkpointing can improve
  - Application performance
  - System utilisation
- RDMA can be used for efficient checkpoint management
- *i*Check – a fully adaptive Invasive Checkpoint Management System

OPENFABRICS ALLIANCE

INVASIVE COMPUTING

INVASIVE COMPUTING

- **DFG Transregio Research Centre TRR89 "Invasive Computing“ – InvasIC**

Deutsche Forschungsgemeinschaft

- **Focus**
  - Dynamic resource management on massively parallel chip multiprocessors
  - Resource-aware applications: invading, infecting, retreating from resources
  - Integration: Hardware - OS – Language & Compiler - Tools – Applications
- **Investigation in the context of HPC – Technical University of Munich**
  - Chair for Scientific Computing: Hans Bungartz, Michael Bader
  - Chair for Computer Architecture and Parallel Systems: Michael Gerndt

*I*CHECK & MALLEABILITY

- **iCheck supports invasive/malleable applications developed using Invasive (Malleable) HPC Infrastructure**

- **Invasive HPC Infrastructure**
  - System level
    - Malleable Resource and Job Management System – ***i*RM**
    - Resource aware MPI – ***i*MPI**
  - Application level
    - Programming models – Elastic Phase Oriented Programming model (EPOP)
    - Applications – Tsunami Simulation, *i*MD, *i*Heat, *i*SWE, ***i*Sum**
  - Services
    - Data analytics application support (Using Apache Spark)
    - Power budget enforcement
    - Fault tolerance - ***i*Check**

INVASIVE INFRASTRUCTURE + *i*CHECK

Invasive Resource Manager (*i*RM)

- Extension of SLURM with dynamic resource management

Invasive Message Passing Interface (*i*MPI)

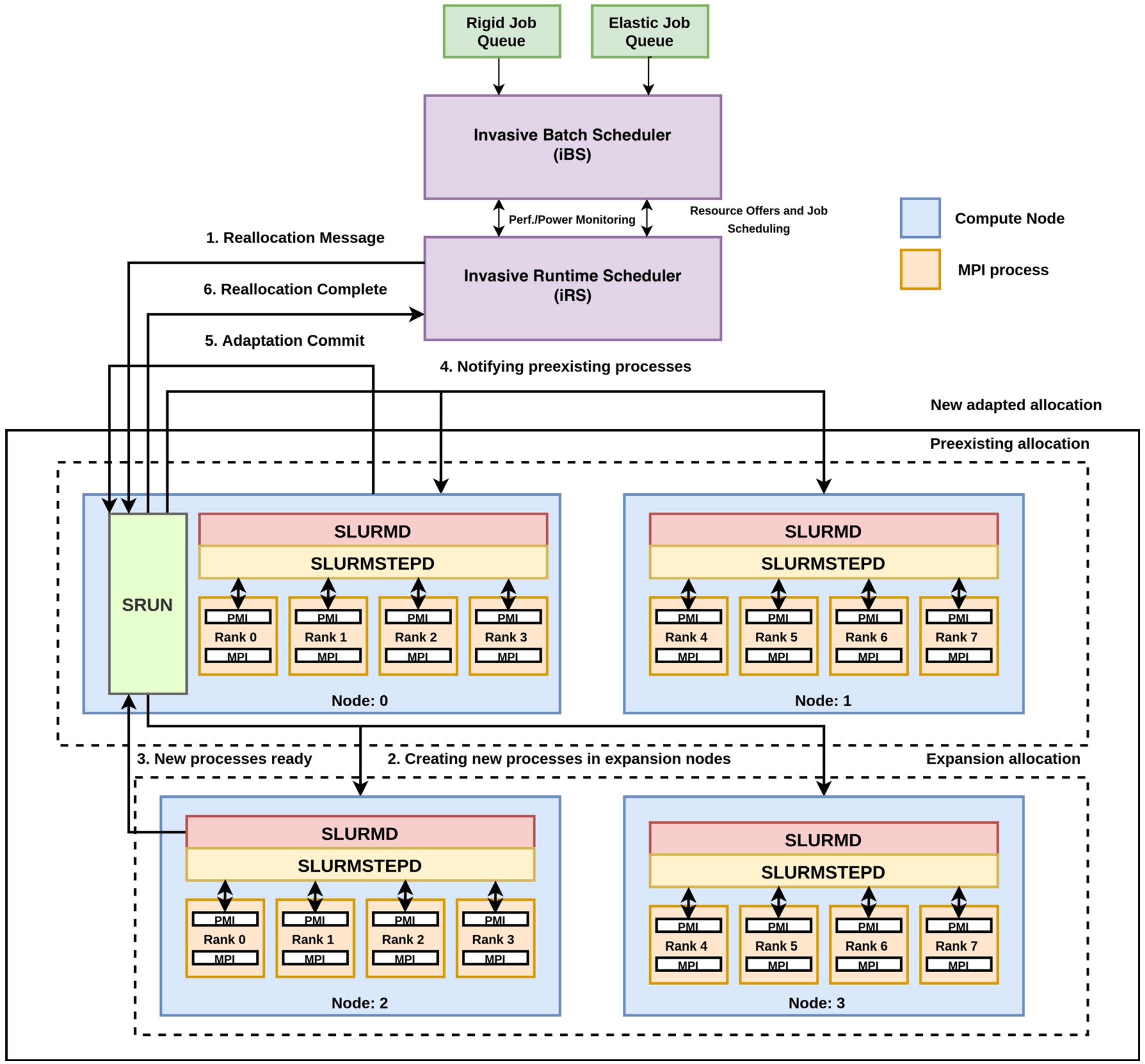
- Extension of MPICH
- Four new operations for dynamic processes
  - *MPI_Init_adapt*
  - *MPI_Probe_adapt*
  - *MPI_Comm_adapt_begin*
  - *MPI_Comm_adapt_commit*

I. A. Comprés Ureña and Michael Gerndt. Towards Elastic Resource Management. In *Tools for High Performance Computing 2017*, pages 105–127. Springer International Publishing, 2019

Ao Mo-Hellenbrand. Resource-Aware and Elastic Parallel Software Development for Distributed-Memory HPC Systems. Dissertation, Technische Universität München, 2019

INVASIVE RESOURCE MANAGER - *i*RM

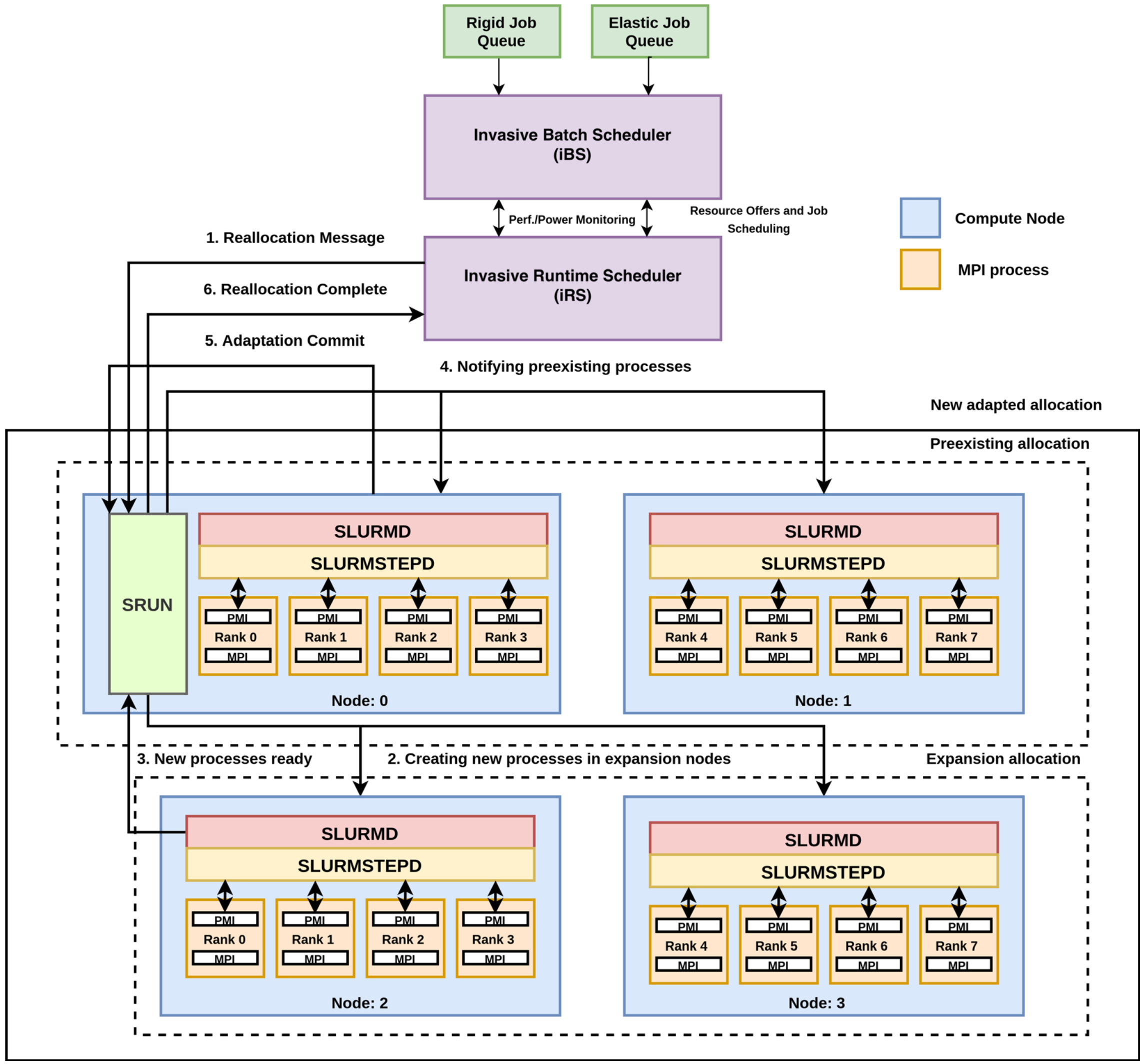
- *i*RM decides about resource reconfiguration
- Application expansion or reduction is possible
- Application reacts to the resource changes triggered by *i*RM
- Resource change is a six-step process



Mohak Chadha, Jophin John, Michael Gerndt. *Extending SLURM for Dynamic Resource-AwareAdaptive Batch Scheduling*. 2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC).

INVASIVE RESOURCE MANAGER - *i*RM

- *i*RM decides about resource reconfiguration
- Application expansion or reduction is possible
- Application reacts to the resource changes triggered by *i*RM
- Resource change is a six-step process
- Scheduler Plugin is created for *i*Check
- *i*Check interacts with the scheduler



Mohak Chadha, Jophin John, Michael Gerndt. *Extending SLURM for Dynamic Resource-AwareAdaptive Batch Scheduling*. 2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC).

INVASIVE MPI - IMPI

MPI_Init_adapt(...)

- Notifies iRM about dynamic application

MPI_Comm_adapt_begin (...)

- Begins the adaptation (start of adaptation window)

MPI_Comm_adapt_commit (...)

- Finalizes the adaptation (end of adaptation window)

MPI_Probe_adapt (...)

- Probes for any resoure changes

Pseudocode - a simple Invasive MPI Application

```
int main() {
 ...
 MPI_Init_adapt(...,mytype)
 // Initialization block
 if mytype == initial_process {
       set phase index = 0
 }
 else {
 // Newly joining processes
       MPI_Comm_adapt_begin (...)
       // Redistribute data
       MPI_Comm_adapt _commit ( ) ;
 }
 // Begin elastic block 1
 if (phase index == 0) {
       while ( block_condition ) {
              MPI_Probe_adapt (...)
              if resource_change {
                   MPI_Comm_adapt_begin (...)
                   // Redistribute data
                   MPI_Comm_adapt _commit ( ) ;
              }
        iteration number++;
        // Compute Intensive part
        }
 }
 // End elastic block 1
 ...
 // Begin elastic block n
 if (phase index == n){ ...
 }
 // End elastic block n
 phase index++;
...
```

Pseudocode - a simple Invasive MPI Application

```
int main() {
 ...
 MPI_Init_adapt(...,mytype)
 // Initialization block
 if mytype == initial_process {
       set phase index = 0
 }
 else {
 // Newly joining processes
       MPI_Comm_adapt_begin (...)
       // Redistribute data
       MPI_Comm_adapt _commit ( ) ;
 }
 // Begin elastic block 1
 if (phase index == 0) {
       while ( block_condition ) {
             MPI_Probe_adapt (...)
             if resource_change {
                MPI_Comm_adapt_begin (...)
                // Redistribute data
                MPI_Comm_adapt _commit ( ) ;
             }
        iteration number++;
        // Compute Intensive part
        }
 }
 // End elastic block 1
 ...
 // Begin elastic block n
 if (phase index == n){ ...
 }
 // End elastic block n
 phase index++;
...
```

Flow of initial set of processes

Pseudocode - a simple Invasive MPI Application

```
int main() {
 ...
 MPI_Init_adapt(...,mytype)
 // Initialization block
 if mytype == initial_process {
       set phase index = 0
 }
 else {
 // Newly joining processes
       MPI_Comm_adapt_begin (...)
       // Redistribute data
       MPI_Comm_adapt _commit ( ) ;
 }
 // Begin elastic block 1
 if (phase index == 0) {
       while ( block_condition ) {
             MPI_Probe_adapt (...)
             if resource_change {
                  MPI_Comm_adapt_begin (...)
                  // Redistribute data
                  MPI_Comm_adapt _commit ( ) ;
             }
        iteration number++;
        // Compute Intensive part
        }
 }
 // End elastic block 1
 ...
 // Begin elastic block n
 if (phase index == n){ ...
 }
 // End elastic block n
 phase index++;
...
```

Flow of newly added set of processes

Pseudocode - a simple Invasive MPI Application

```
int main() {
 ...
 MPI_Init_adapt(...,mytype)
 // Initialization block
 if mytype == initial_process {
       set phase index = 0
 }
 else {
 // Newly joining processes
       MPI_Comm_adapt_begin (...)
       // Redistribute data
       MPI_Comm_adapt _commit ( ) ;
 }
 // Begin elastic block 1
 if (phase index == 0) {
       while ( block_condition ) {
            MPI_Probe_adapt (...)
            if resource_change {
                 MPI_Comm_adapt_begin (...)
                 // Redistribute data
                 MPI_Comm_adapt _commit ( ) ;
            }
       iteration number++;
       // Compute Intensive part
       }
 }
 // End elastic block 1
 ...
 // Begin elastic block n
 if (phase index == n){ ...
 }
 // End elastic block n
 phase index++;
...
```

Flow of initial set of processes

Pseudocode - a simple Invasive MPI Application

```
int main() {
 ...
 MPI_Init_adapt(...,mytype)
 // Initialization block
 if mytype == initial_process {
       set phase index = 0
 }
 else {
 // Newly joining processes
       MPI_Comm_adapt_begin (...)
       // Redistribute data
       MPI_Comm_adapt _commit ( ) ;
 }
 // Begin elastic block 1
 if (phase index == 0) {
       while ( block_condition ) {
             MPI_Probe_adapt (...)
             if resource_change {
                   MPI_Comm_adapt_begin (...)
                   // Redistribute data
                   MPI_Comm_adapt _commit ( ) ;
             }
       iteration number++;
       // Compute Intensive part
       }
 }
 // End elastic block 1
 ...
 // Begin elastic block n
 if (phase index == n){ ...
 }
 // End elastic block n
 phase index++;
...
```

Flow of initial set of processes

Pseudocode - a simple Invasive MPI Application

```
int main() {
 ...
 MPI_Init_adapt(...,mytype)
 // Initialization block
 if mytype == initial_process {
       set phase index = 0
 }
 else {
 // Newly joining processes
       MPI_Comm_adapt_begin (...)
       // Redistribute data
       MPI_Comm_adapt _commit ( ) ;
 }
 // Begin elastic block 1
 if (phase index == 0) {
       while ( block_condition ) {
             MPI_Probe_adapt (...)
             if resource_change {
                   MPI_Comm_adapt_begin (...)
                   // Redistribute data
                   MPI_Comm_adapt _commit ( ) ;
             }
        iteration number++;
        // Compute Intensive part
        }
 }
 // End elastic block 1
 ...
 // Begin elastic block n
 if (phase index == n){ ...
 }
 // End elastic block n
 phase index++;
...
```

Joining processes

Existing processes

iMPI

Pseudocode
Invasive MPI+iCHECK

```
int main() {
 ...
 MPI_Init_adapt(...,mytype)
 // Initialization block
 if mytype == initial_process {
      set phase index = 0
 }
 else {
 // Newly joining processes
      MPI_Comm_adapt_begin (...)
      icheck_redistribute();
      MPI_Comm_adapt _commit ( ) ;
 }
 // Begin elastic block 1
 if (phase index == 0) {
      while ( block_condition ) {
           MPI_Probe_adapt (...)
           if resource_change {
                MPI_Comm_adapt_begin (...)
                icheck_redistribute();
                MPI_Comm_adapt _commit ( ) ;
           }
      iteration number++;
      // Compute Intensive part
      }
 }
 // End elastic block 1
 ...
 // Begin elastic block n
 if (phase index == n){ ...
 }
 // End elastic block n
 phase index++;
...
```

Joining processes

Existing processes

Pseudocode - a simple Invasive MPI Application

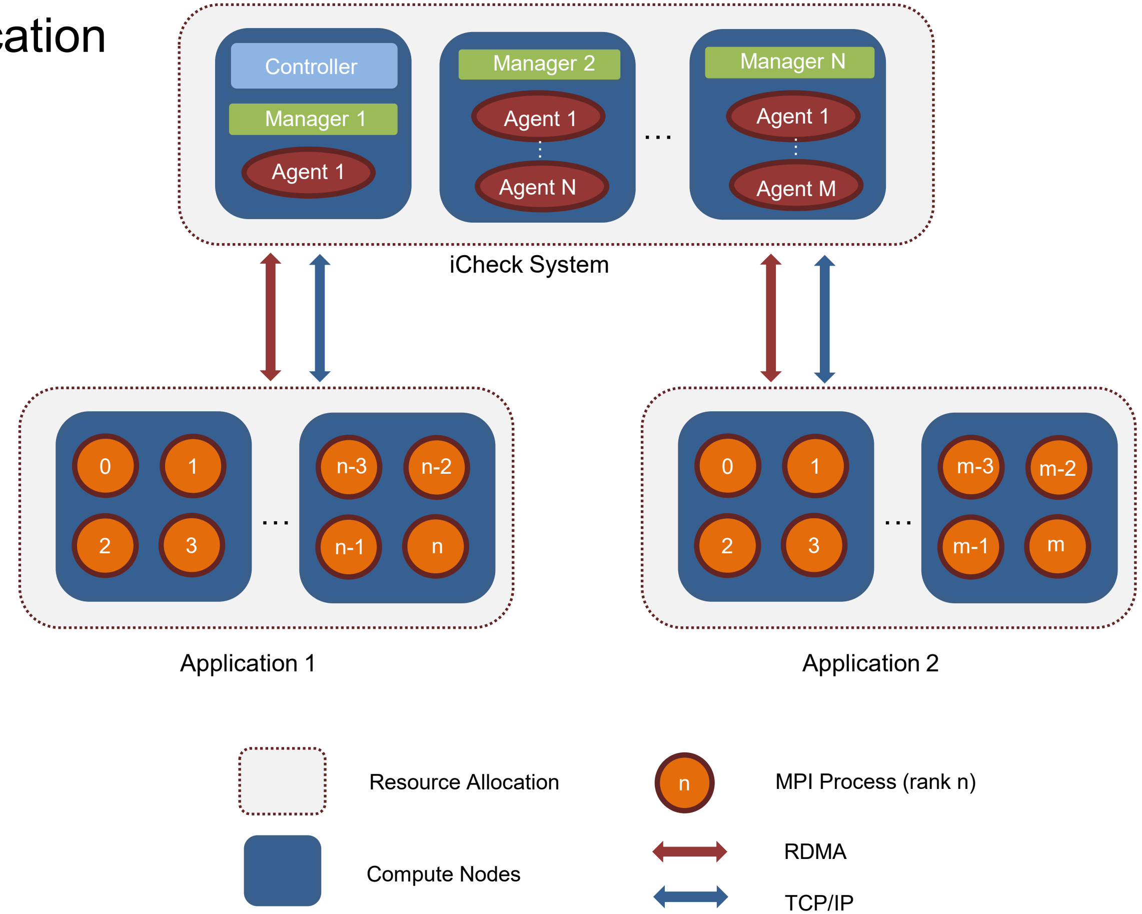
```
int main() {
 ...
 MPI_Init_adapt(...,mytype)
 // Initialization block
 if mytype == initial_process {
       set phase index = 0
 }
 else {
 // Newly joining processes
       MPI_Comm_adapt_begin (...)
       // Redistribute data
       MPI_Comm_adapt _commit ( ) ;
 }
 // Begin elastic block 1
 if (phase index == 0) {
       while ( block_condition ) {
            MPI_Probe_adapt (...)
            if resource_change {
                MPI_Comm_adapt_begin (...)
                // Redistribute data
                MPI_Comm_adapt _commit ( ) ;
            }
        iteration number++;
        // Compute Intensive part
        }
 }
 // End elastic block 1
 ...
 // Begin elastic block n
 if (phase index == n){ ...
 }
 // End elastic block n
 phase index++;
...
```

All processes starts working together

/CHECK

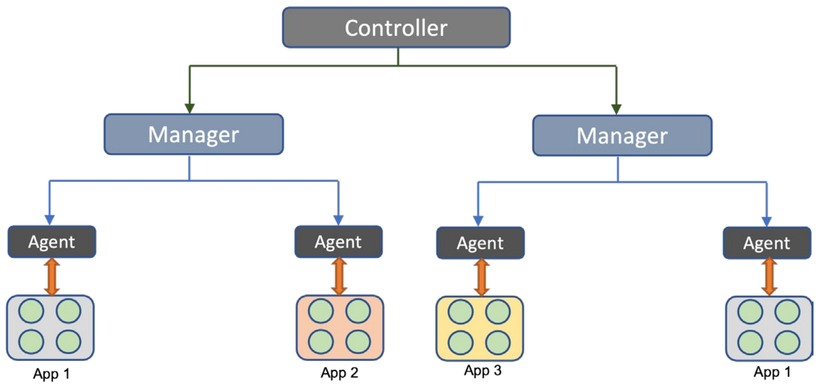
*I*CHECK ARCHITECTURE

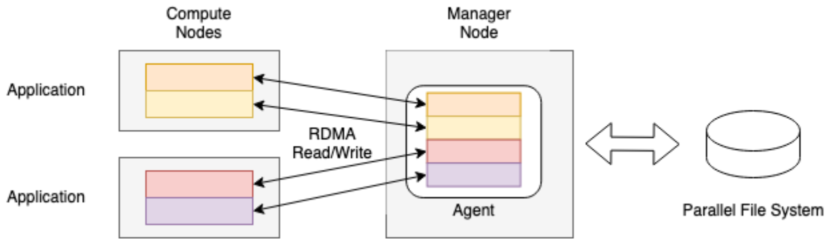
- *i*Check – An invasive RDMA based multilevel application level checkpointing system
- Deployed in dedicated nodes
- *i*Check Core
  - *Controller – single instance component*
  - *Manager – one instance per* i*Check node*
  - *Agents – multiple instances per* i*Check node*
- *i*Check Library
  - Application interacts with iCheck Core
- Simultaneous checkpoint management of multiple applications
  - Potential offered by dynamism is enormous



iCHECK WORKFLOW

- Application registers with the controller
- Controller assigns agents and *i*Check nodes
  - Agent placement algorithm
  - Node selection algorithm
- Manager launches agents
- Agents connect with the applications
- RDMA configuration performed
- Application calls commit & continues the execution
  - Agent retrieves the checkpoint using RMA
- Application can probe for agent change

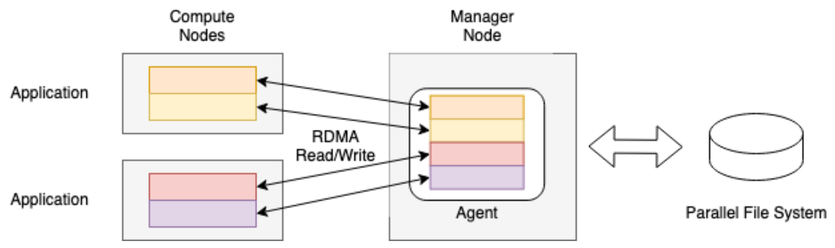




Checkpoint transfer

RDMA IN *I*CHECK

- Application registers memory regions
- Uses Libfabric library
- Multiple techniques in *i*Check
  - RDMA only using memory regions in libfabric
  - RDMA + Shared memory
- Push and Pull Techniques
  - Push: Application pushes checkpoint data to agents
  - Pull: Agents read checkpoint data from applications
- Agents write data to PFS

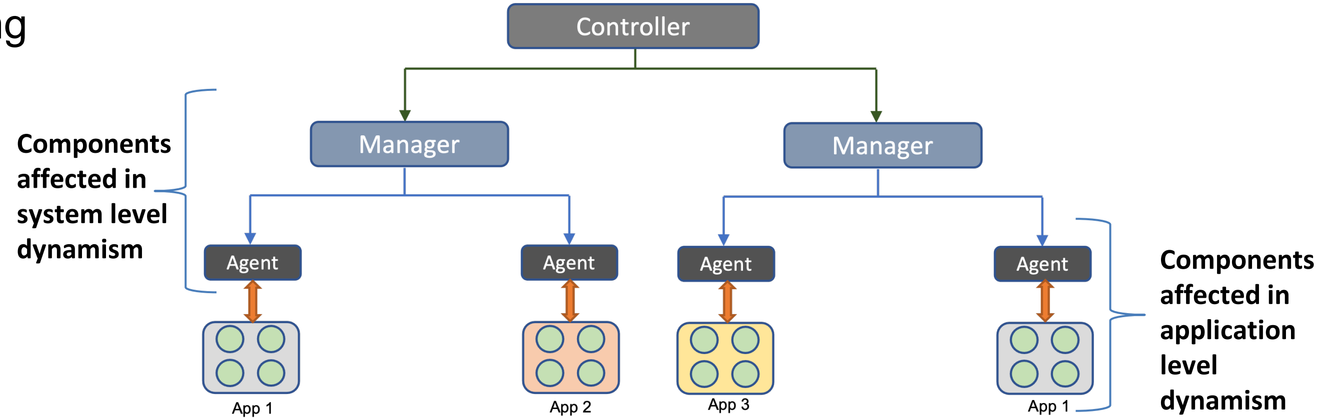


Checkpoint transfer

DYNAMISM IN ICHECK

DYNAMISM IN *i*CHECK

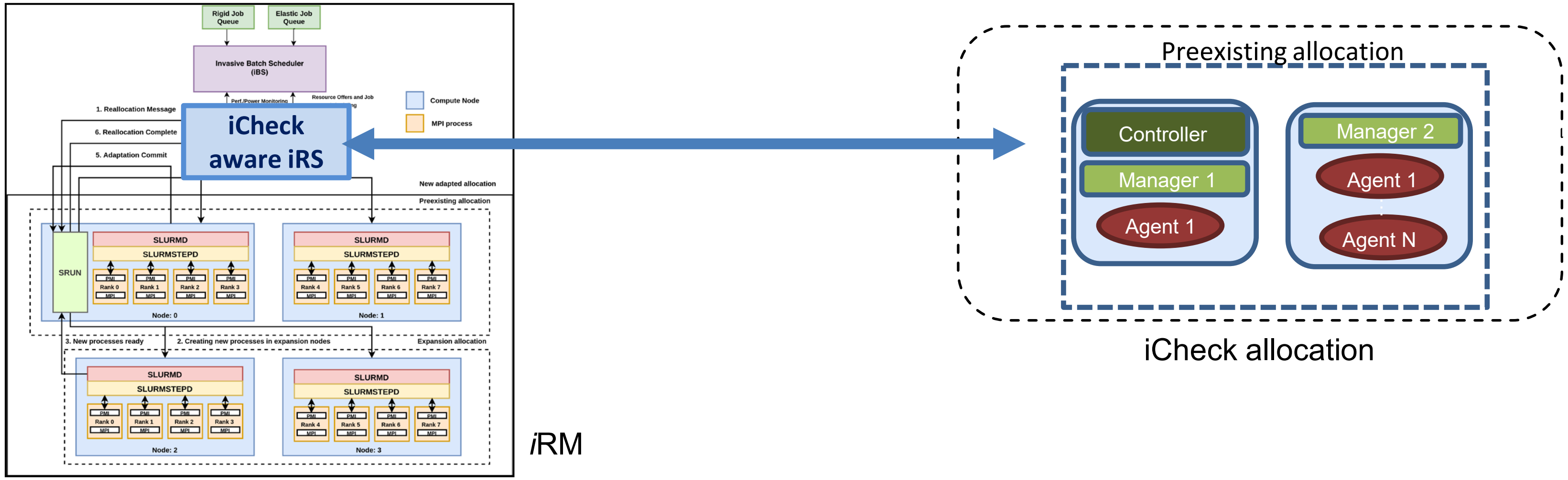
- System-level dynamism in *i*Check
  - Scaling of *iCheck nodes* (Manager) using ***i*RM**
  - *Agent* behaviour
- Application-level dynamism
  - Scaling of *Agents*
  - Support for *Malleability*



Hierarchical view of *i*Check system

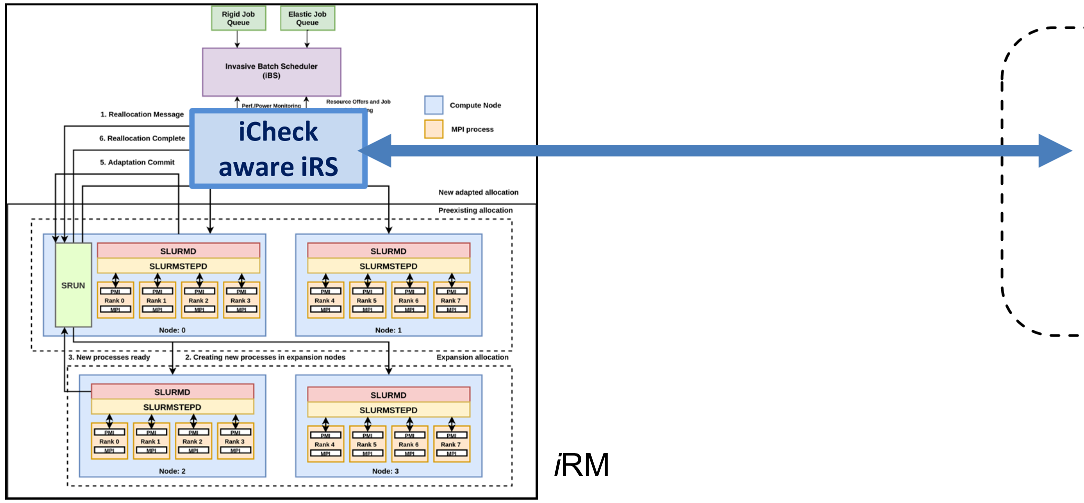
SYSTEM-LEVEL DYNAMISM – *ICHECK NODES*

- Created a new Slurm plugin to support *i*Check
- iRM can reconfigure *i*Check nodes (Naive approach)
  - *i*Check can also request for new nodes

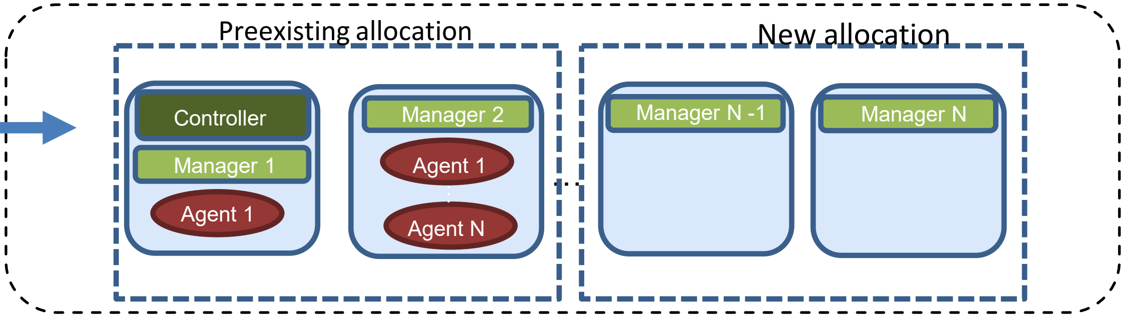


SYSTEM-LEVEL DYNAMISM – *ICHECK NODES*

- Created a new Slurm plugin to support *i*Check
- iRM can reconfigure *i*Check nodes (Naive approach)
  - *i*Check can also request for new nodes
- Complex strategies are necessary



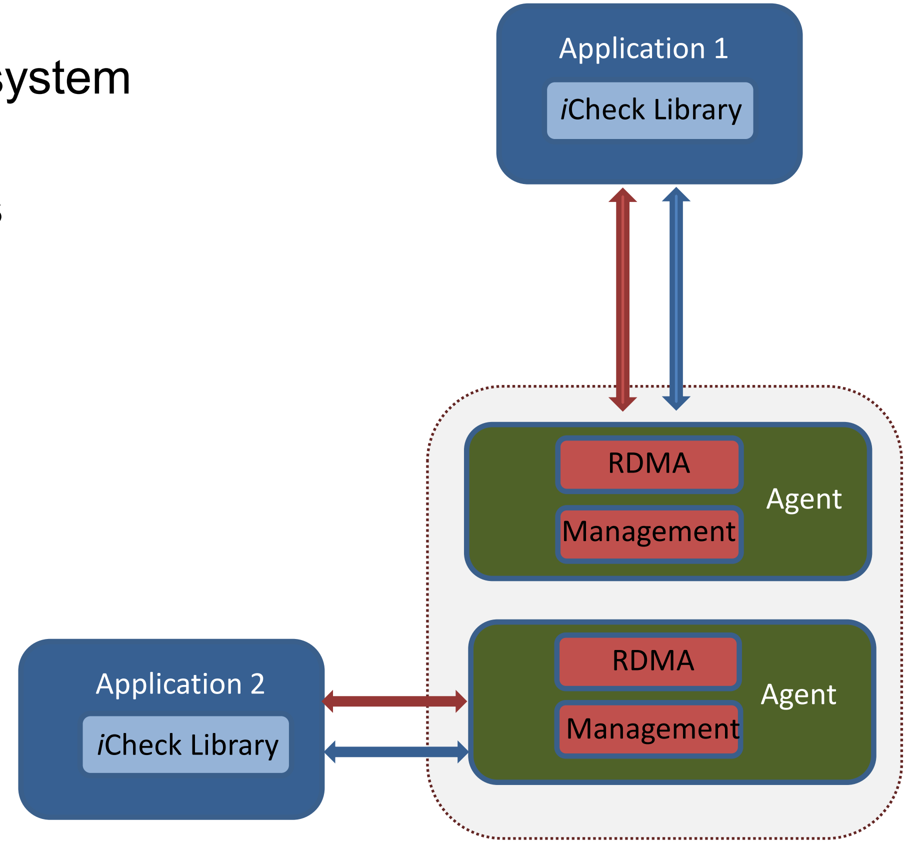
*i*RM



iCheck allocation

SYSTEM-LEVEL DYNAMISM - *AGENTS*

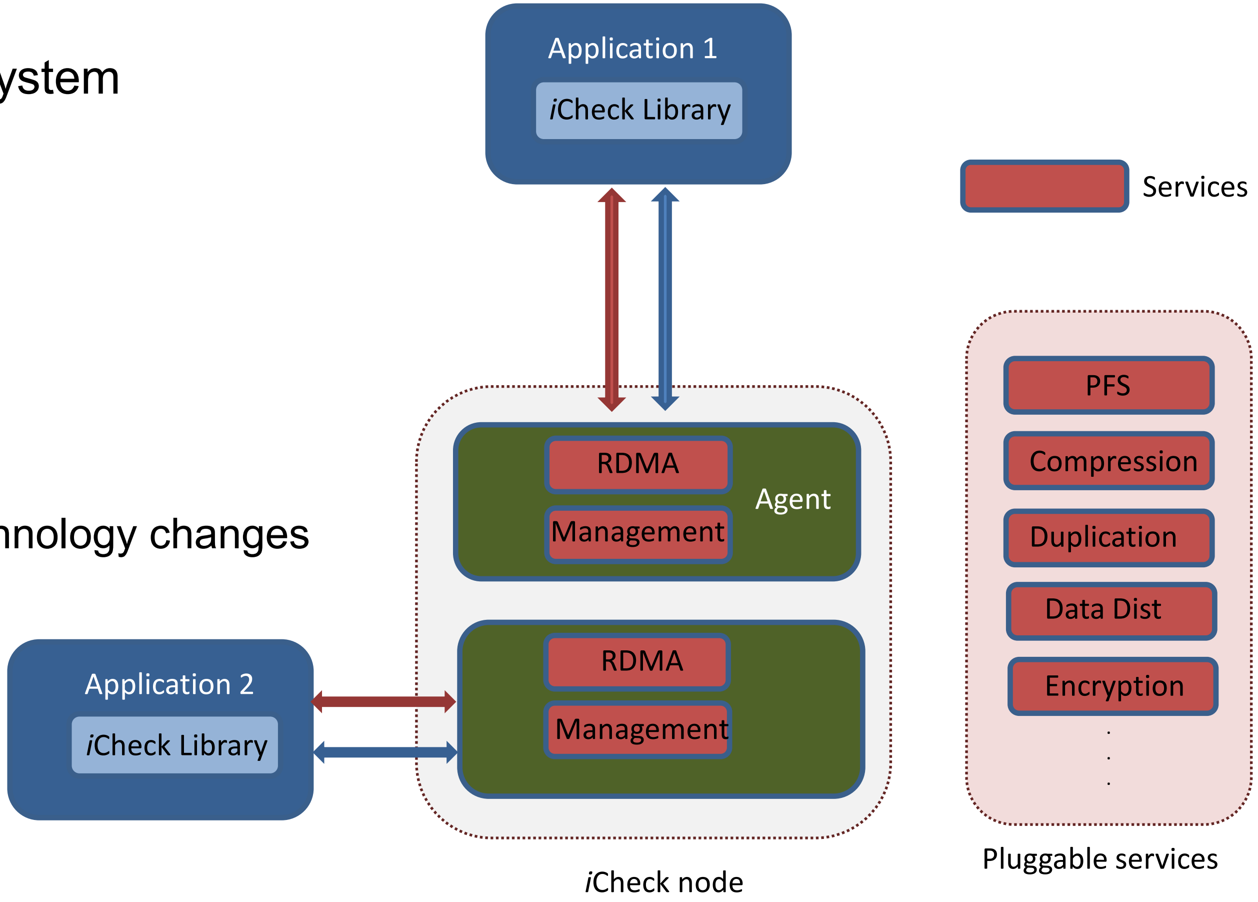
- Agents are lowest level component in *i*Check system
- Agents write/read checkpoint from applications
- Agents can do more than checkpoint retrieval



*i*Check node

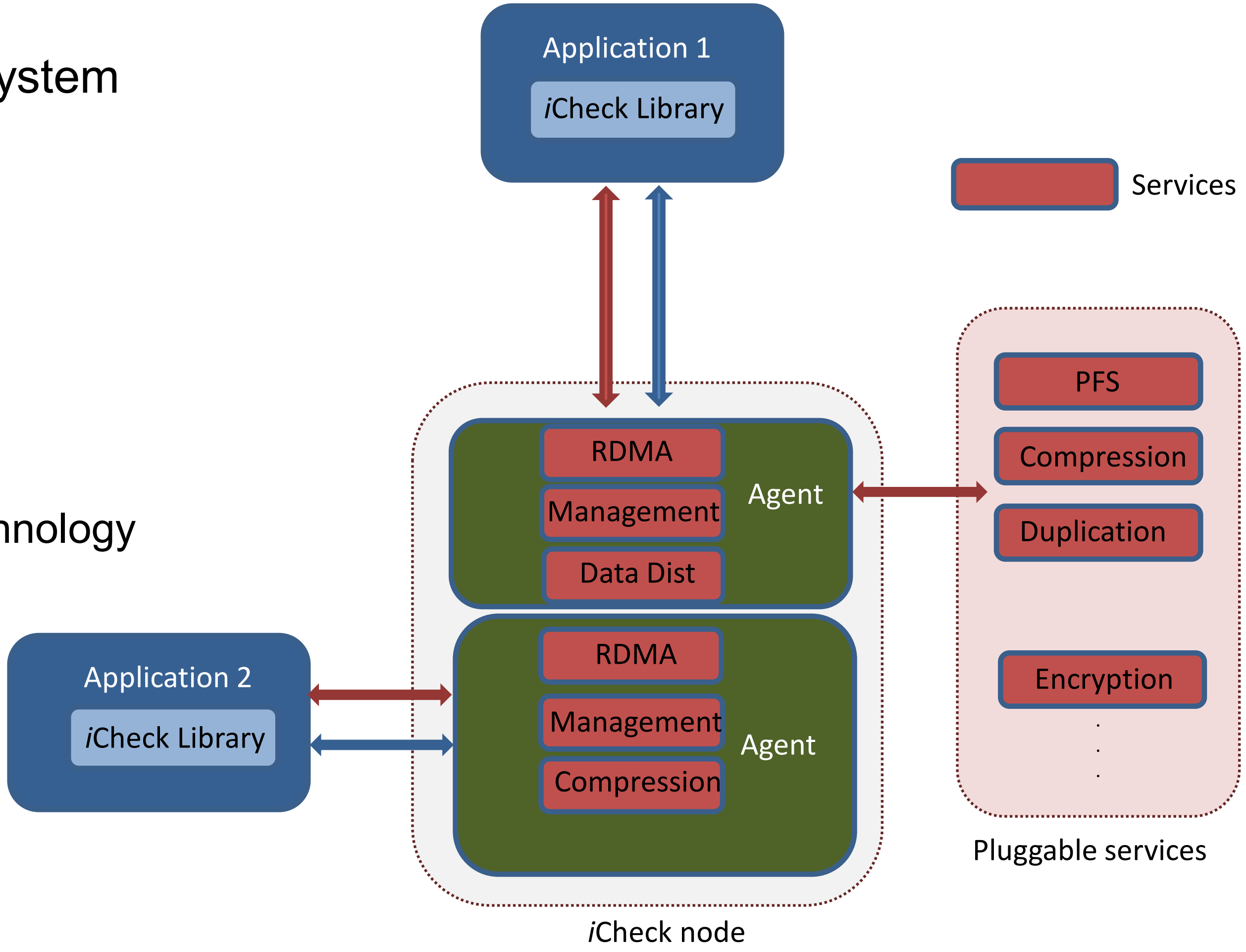
SYSTEM-LEVEL DYNAMISM - *AGENTS*

- Agents are lowest level component in *i*Check system
- Agents write/read checkpoint from applications
- Agents can do more than checkpoint retrieval
- iCheck offers different plugins
  - New plugins can be added as requirement and technology changes
  - Does not need to worry about optimisation



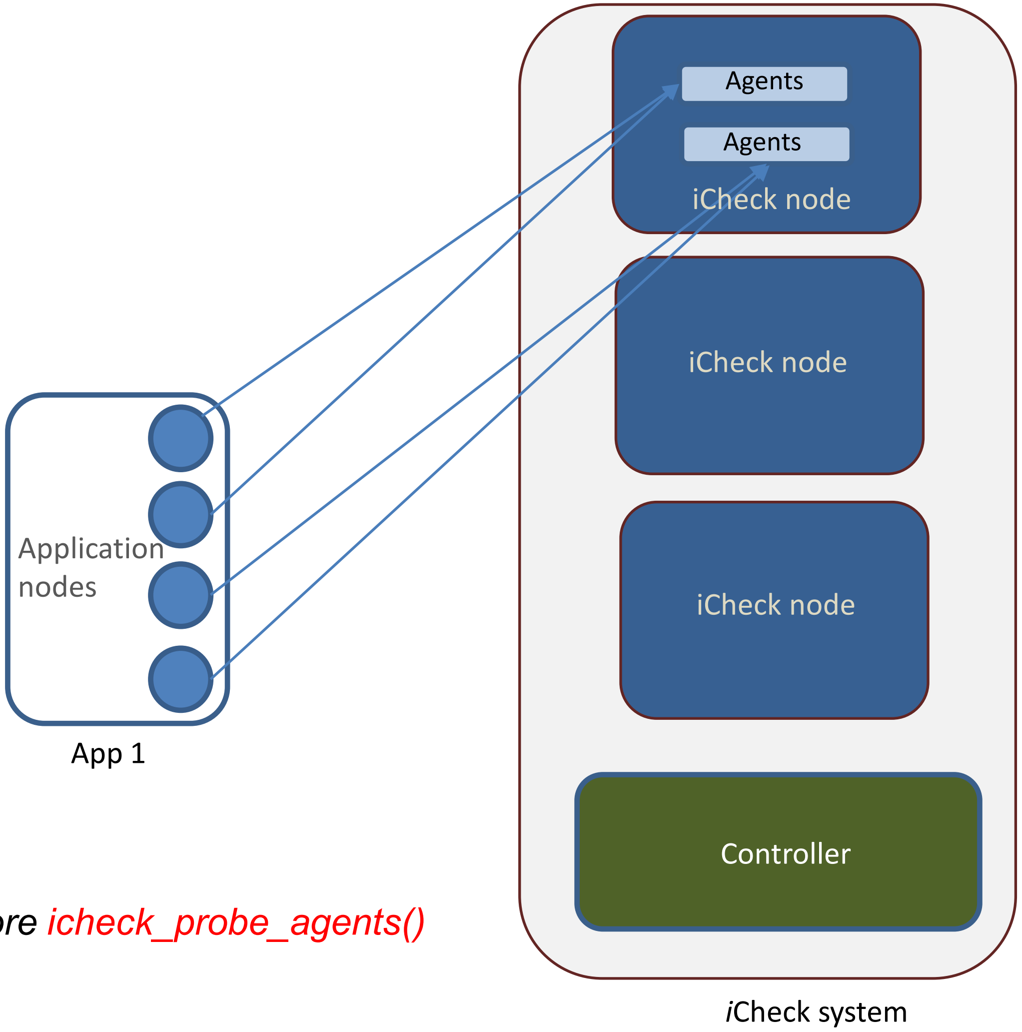
SYSTEM-LEVEL DYNAMISM - *AGENTS*

- Agents are lowest level component in *i*Check system
- Agents write/read checkpoint from applications
- Agents can do more than checkpoint retrieval
- iCheck offers different plugins
  - New plugins can be added as requirement and technology changes
  - Does not need to worry about optimisation
- Checkpointing systems can do a lot more on-the-fly



APPLICATION-LEVEL DYNAMISM - *AGENTS*

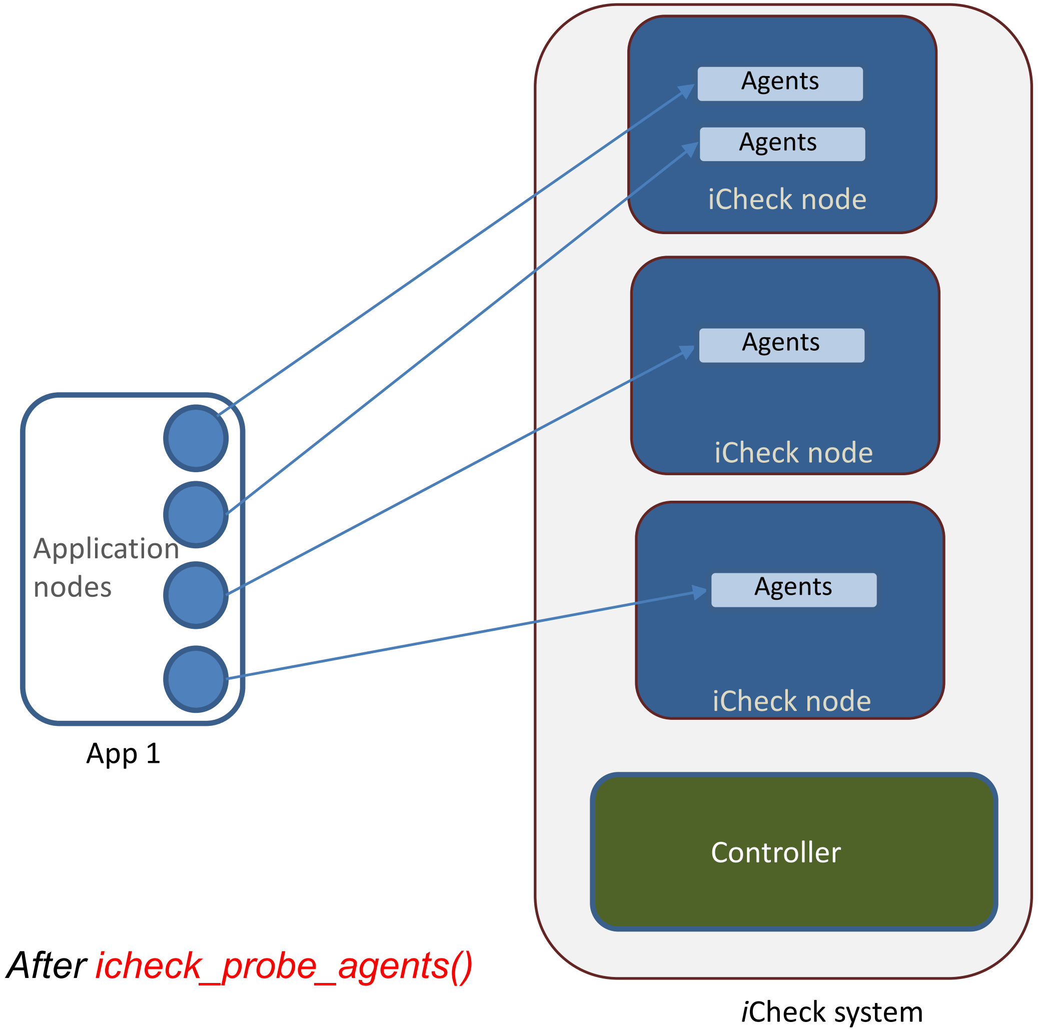
- Basic API
  - *icheck_init*(*“appname", …, **status***);
  - *icheck_add*(*“var_name", &var, **SIZE***);
  - *icheck_commit ( ) ;*
  - *icheck_restart ( ) ;*
  - *icheck_restore*(*var_name,&var*)
  - *icheck_finalize (icheck state) ;*
  - *icheck_enable_async*() – enable asynchronous checkpointing
- API call *icheck_probe_agents()* will contact controller
- RDMA reconfiguration
- Controller can decide on agent change



Before icheck_probe_agents()

APPLICATION-LEVEL DYNAMISM - *AGENTS*

- Basic API
  - *icheck_init(“appname", …, **status**);*
  - *icheck_add(“var_name", &var, **SIZE**);*
  - *icheck_commit ( ) ;*
  - *icheck_restart ( ) ;*
  - *icheck_restore(var_name,&var)*
  - *icheck_finalize (icheck state) ;*
  - *icheck_enable_async*() – enable asynchronous checkpointing
- API call *icheck_probe_agents()* will contact controller
- Controller can decide on agent change
- RDMA reconfiguration
- Performs agent change on-the-fly based on the application metrics



After icheck_probe_agents()

APPLICATION-LEVEL DYNAMISM - *MALLEABILITY*

- **Support** checkpoint restart in malleable applications
- Perform data redistribution for malleable applications
- Malleable API
  - *icheck_add_adapt(“var_name", &var, sizeof(int),…, **TYPE**);*
  - *icheck_redistribute(var_name,&var,.., **TYPE**)*

```
#include<icheck.h>
int main() {
 MPI_Init_adapt(..., type)
 float data[SIZE];
 icheck_init(..., type);
 icheck_add_adapt("data",data,...,BLOCK);
 if(checkpoint_available && no_adapt){
   icheck_restart();
 }
 if (type == joining) {
     MPI_Comm_adapt_begin(...);
     icheck_redistribute("data",data,
               new_size, BLOCK)
     MPI_Comm_adapt_commit();
 }
 while (true){
    MPI_Probe_adapt(resource_change,...);
    if (resource_change) {
        MPI_Comm_adapt_begin(...);
        icheck_redistribute("data",data,
               new_size, BLOCK)
        MPI_Comm_adapt_commit();
    }
    /*Read/Write data[]*/
    if(iteration%100)
        icheck_commit();
    /*Check for agent change*/
    if(iteration%1000)
        icheck_probe_agents();
    }
    icheck_finalize(IC_PERSIST);
    MPI_Finalize();
}
```

Pseudocode using iCheck API

EVALUATION

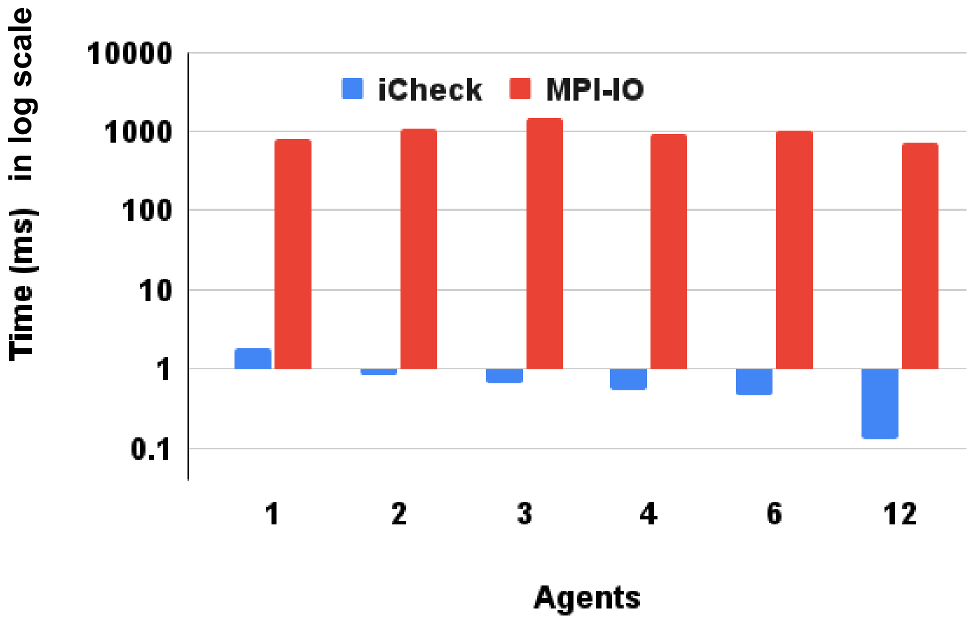
EVALUATION SETUP

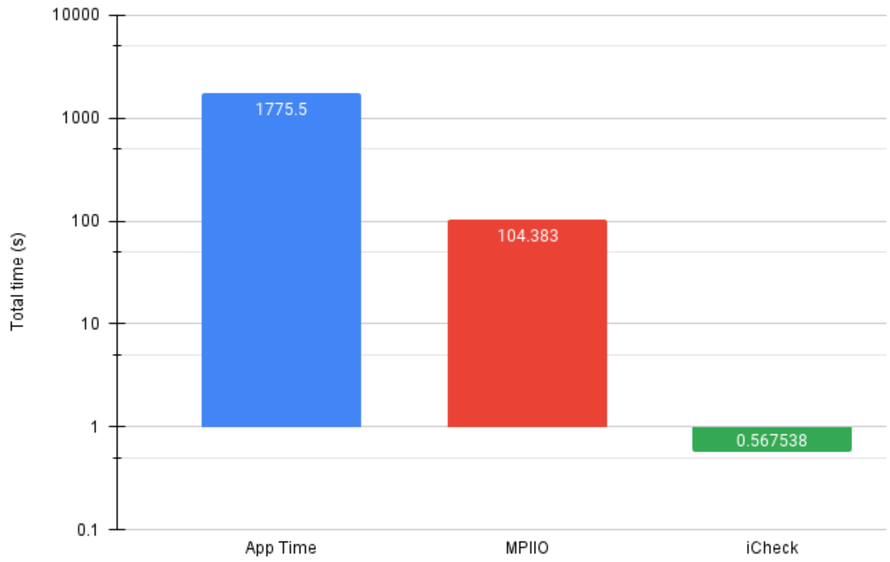
- Tests were performed on SuperMUC-NG
- Evaluation on 16 nodes (768 cores)
  - 12 nodes (576 cores) for applications
  - Four nodes for iCheck
- Four applications
  - ls1 mardyn
  - LULESH
  - 2D Jacobi heat simulation
  - Synthetic application – checkpoints 2.3GB per second - 1million floats
- Scenarios:
  - Compare *i*Check vs MPI-IO in ls1 mardyn
  - Compare *i*Check vs MPI-IO vs SCR in a synthetic application
  - Effect of dynamic agents on a 2D heat simulation application
  - Effect of different agent placement strategies using multiple synthetic applications

SuperMUC-NG at LRZ, Germany
https://doku.lrz.de/display/PUBLIC/SuperMUC-NG

RESULTS

iCheck vs MPI-IO in ls1 mardyn



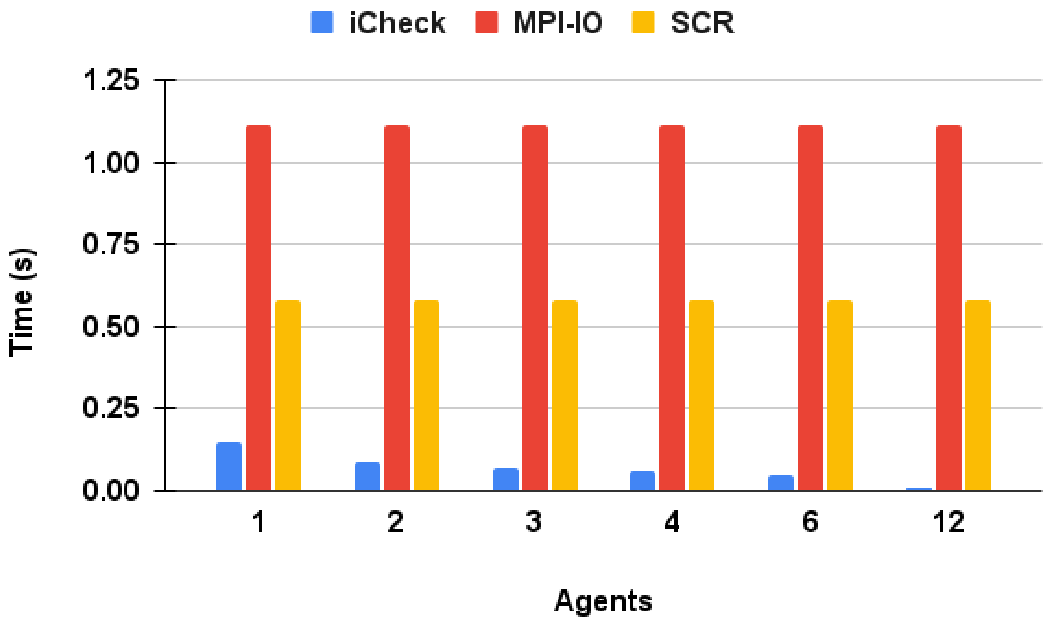


Up to 5000 times (best case with 12 agents) faster checkpointing with iCheck.

In the worst case, iCheck is 400 times faster (single agent).

RESULTS

iCheck vs MPI-IO vs SCR in Synthetic application

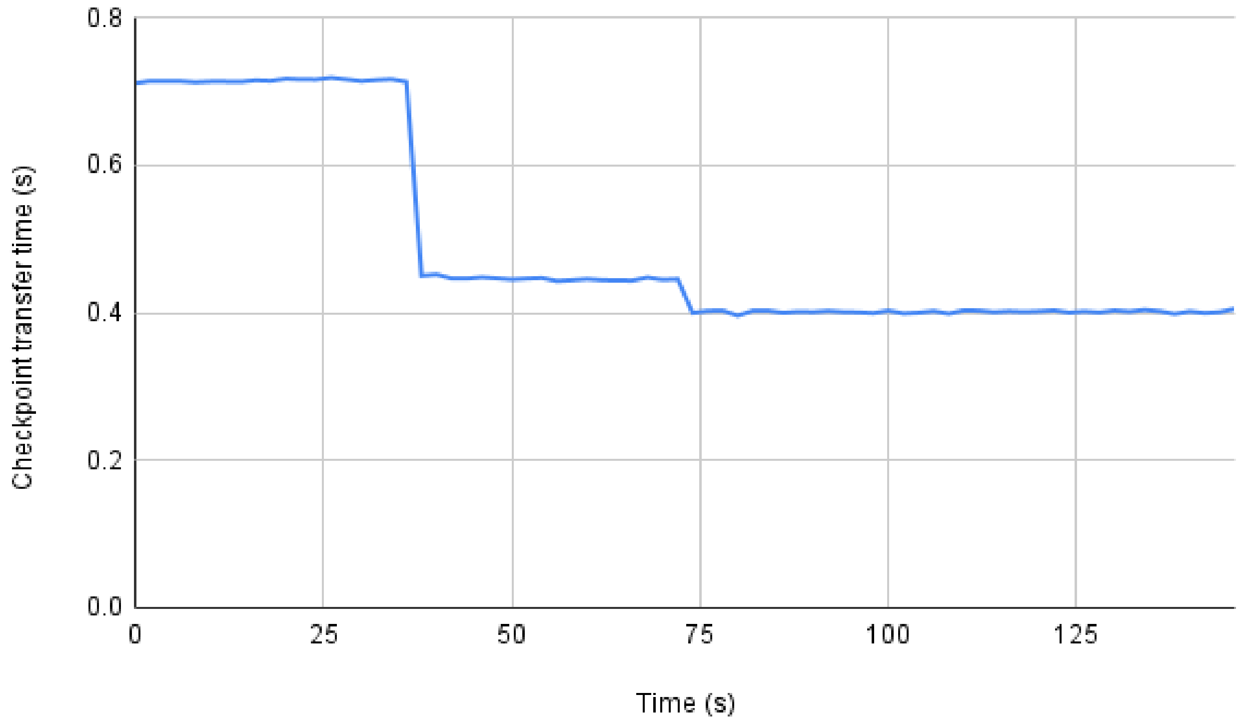


Up to 100x and 57x faster checkpointing (using 12 agents) than MPI-IO and SCR

In the worst case (using single agent), iCheck is 8x faster than MPI-IO and 4x faster than SCR

RESULTS

Effect of dynamic agents on a 2D heat simulation application

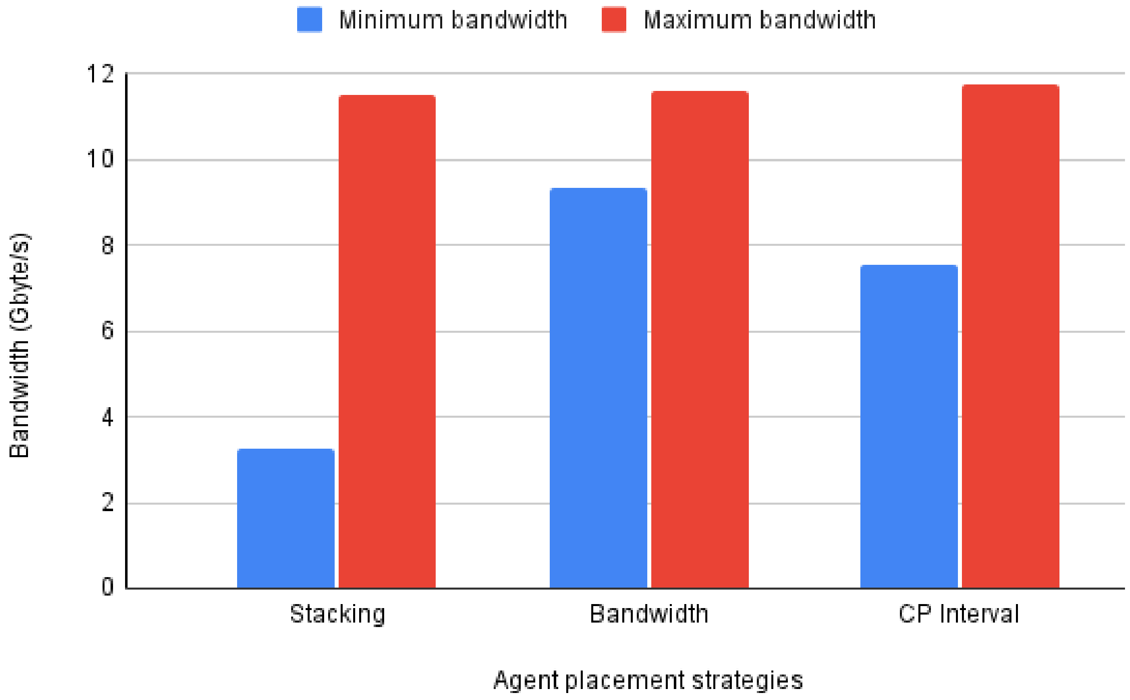


- Dynamism might not be ideal in all cases
- Heuristics are needed to model the performance characteristics

Up to 50 times faster performance possible on the fly

RESULTS

Effect of different agent placement strategies in synthetic applications



- Stacking: all agents in same iCheck node
- Bandwidth: placed agents based on available bandwidth
- CP Interval: placed agents on checkpoint interval

Significant improvement in bandwidth utilisation with ideal agent placement

CHALLENGES

Data redistribution

Redistribution not trivial

Complex datatypes

Checkpoint management

Agent placement

Efficient resource utilisation

Privacy and Data

Multiple applications

IO operations

INTO THE FUTURE

Optimisation potential	Pluggable services Hardware
Data & Resource management	Malleable Non-Malleable
Deployment in Cloud	HPC on Cloud iCheck can be deployed easily
Policies for checkpoint management (not limited to performance improvement)	Power aware Carbon aware
Support for Accelerators	GPUS FPGA

2023 OFA Virtual Workshop

THANK YOU

Jophin John

Chair of Computer Architecture and Parallel Systems

Technical University of Munich, Germany