# Remote process execution in idle workstation

## Introduction

A Study by Matt and Miron[1] states that workstations in academic institutions and workplaces are under-utilized. Their observation in a network of workstations showed they were idle 70% of the time. Even in the busiest times, a considerable amount of CPU cycles were getting wasted. Various kinds of literature were proposed to harness the wasted cycles. One such idea is to remotely execute the processes in idle workstations and be transparent to the program and user. In this system, computation is done in the idle workstation and I/O in the user's workstation[Maintains the transparency]. Questions might arise regarding the efficiency of executing programs remotely. Studies have shown that remote execution has better CPU time than local execution.

Papers regarding remote execution, arrange their content in three main topics. They are the **remote execution environment, Process scheduling,** and **Fault-tolerance**. Content in this review compares and evaluates remote environment and fault-tolerance aspects from papers like Remote unix[2], Butler system[3], Condor[4], and DAWGS[5]. In addition to that, this review also provides a new fault-tolerant mechanism against the conventional checkpoint and restore mechanism. This new method is more efficient concerning the storage, network bandwidth consumption, and execution time than the latter.

In this review, the term '*issue node*' or '*issue workstation*' refers to the user's workstation that issues a job/process. '*Remote node*' refers to the node which does the computation of the job issued by the user. '*System' or 'middleware*' refers to the setup/application that provides the remote execution environment, scheduler, and fault-tolerance.

## Remote Execution Environment

The objective of this module is to provide a local environment in the remote node. All papers follow a similar setup to simulate the local environment as shown in Fig1. Syscalls made by remotely executing processes get reverted to the issue node. A shadow process in the issue node executes those syscalls and sends back the results. This redirection of I/O syscalls provides an environment for remote execution as of the local environment.

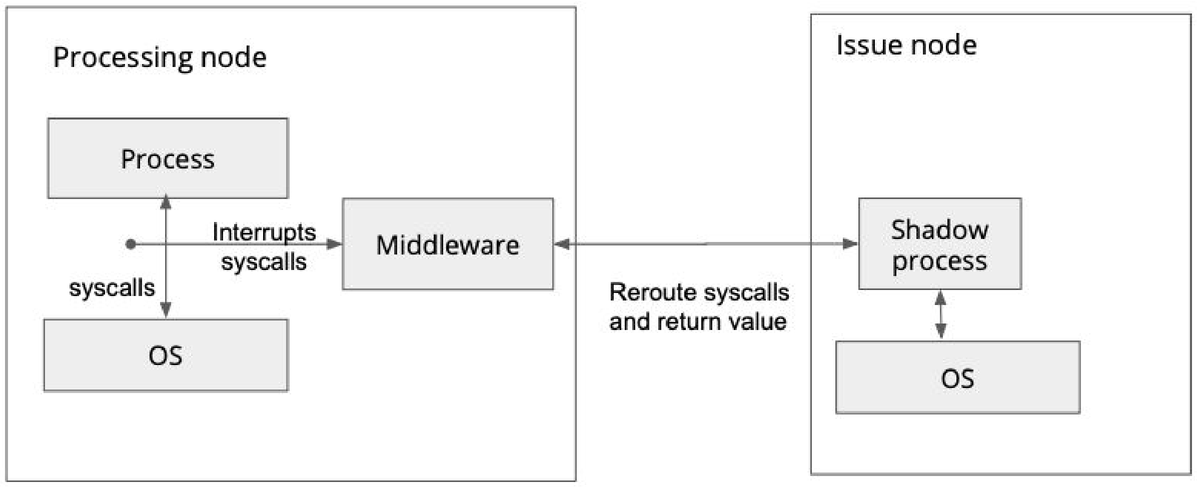


Fig 1: General Remote process execution paradigm

The papers differ in terms of the degree of transparency, I/O supported, and prerequisite of the environment. Systems like DAWGS provide complete transparency and support standardIO and data files. Whereas Condor and Remote Unix expect users to link their libraries with the program and Condor limits its support to data files alone, Remote Unix expects users to provide files for standardIO. The butler system supports GUI programs but requires Andrew File System[6] and special window server for functioning.

The limitations in these papers are, they expect a homogeneous setup with UNIX and don't support programs that spawn child processes, use IPCs and SocketIO. Those features could be enabled as well with necessary syscall redirection and bookkeeping. By enabling child process and IPC support, parallelism in programs could be exploited as well. By assigning a separate idle machine to the spawned child and parent, process execution will be faster. The parallelism in threads can not be exploited in such a way, since they communicate using shared memory.

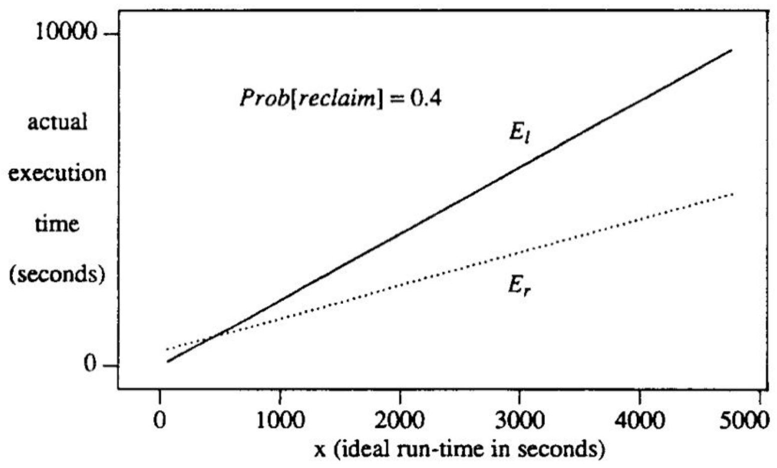


Fig 2: Taken from DAWGS paper. El & Er represents local and remote execution respectively. At fixed failure or reclaim probability [Seen in the next module] of 0.4 above execution pattern is observed. Larger the actual execution time, the better the remote execution. The point where the remote execution advances local execution is called the '*break-in point*'.

Though remote execution has an overhead of syscall redirection and initial process migration. It still has better execution time than local execution  because of the load factor. Local machines might have more load, and the frequency of CPU time quantum allowed for a process to execute is less compared to load free remote machines(Idle). Hence, remote execution completes faster. These results are better observed when the process has higher ideal run-time as seen in Fig2.

## Fault-tolerance

These systems prioritize the user of the idle workstation. On his/her return the remote process executing on that node should be suspended or moved to another idle machine. The system requires a mechanism for handling workstation crashes and restarts as well. All papers except the butler system use checkpointing as a fault tolerance mechanism. When a remote process executes, the middleware facilitates the capture of process state[stack, register values, files opened, etc] at regular intervals. If the process terminates, the latest checkpoint is restored on another machine. This mechanism ensures the eventual completion of the jobs.

The time taken for remote execution in this method is **Tr = X+Sh+M+R+(x//a)C+F**. '*X*' refers to actual process execution time, '*Sh*' stands for the time taken for finding idle nodes, '*M*' for initial process migration, *'R'* for returning the execution results, *'C'* is checkpointing overhead. Since checkpoints are taken in *'a'* time interval, so x//a checkpoints in total execution. '*F*' is the time required for process migration after failure. This method also has an overhead of redoing the computation. In case of a process failure, restarting the process from the previous checkpoint will make the process redo computation that might last around [0, a] time quantum(DAWGS captures checkpoints every 30min, so if the process fails shy of the next checkpoint, redo of 30min should be done). Besides time overhead storage and network bandwidth are consumed. Systems like DAWGS move the checkpoint data to the issue node(consumes network bandwidth). Whereas Condor saves the checkpoint in the remote machine itself, only transferred if process migration is required, but this is not resilient against remote node crashes.

Though checkpoint is an effective method for fault-tolerance, it also frustrates users who issue an interactive job because of redo and time lag for process migration after failure. Arguments may arise that more users of this system issue compute-bound processes. But the butler paper's report of jobs for a month states that a considerable amount of submitted jobs were interactive. Applications that were memory intensive were executed remotely. A better method could be proposed to overcome the Checkpoint’s shortcomings.

***Active replication method***

*If the system could replicate the process in multiple machines and make sure it appears as a single process to the user, then fault-tolerance could be achieved with less overhead in the network, storage, and time.* This might consume more computing power, but it's better to spend computing power than time, network, and storage.

When a user exports a process, it is made to execute in multiple idle systems simultaneously(Clones). Computation of the clones could proceed independently. On a system call, they are reverted to the issue node and the shadow program makes sure that a system call is executed at most once. Each system call is logically timestamped[7] by the respective remote node’s middleware and sent to the issue node for execution. On receiving the system call the shadow process inspects the timestamp. If new, it executes, caches along with the timestamp, and replies with the return value. If not, it returns the cached value for the respective timestamp. This at most once system call implementation makes the multiple execution transparent to the user.

In case the user returns to a remote node, the process clone executing in that node could be suspended or killed. This action won't affect other clones, they continue executing without any interruptions. The user experiences very little to no delay in the execution. This method provides faster execution since now **Tr = X+Sh+M+R** only and also eliminates time delay because of re-execution in case of failures. Users get a better experience if he/she exports interactive processes. Fig3 taken from the DAWGS paper shows job execution of local vs remote, with the x-axis varying the probability of the user reclaiming the idle workstation. We could see the remote execution time increase exponentially with more process migration.

Active replication strategy will *'almost'* have linear execution irrespective of the probability of reclaim [Changes if all clones fail].

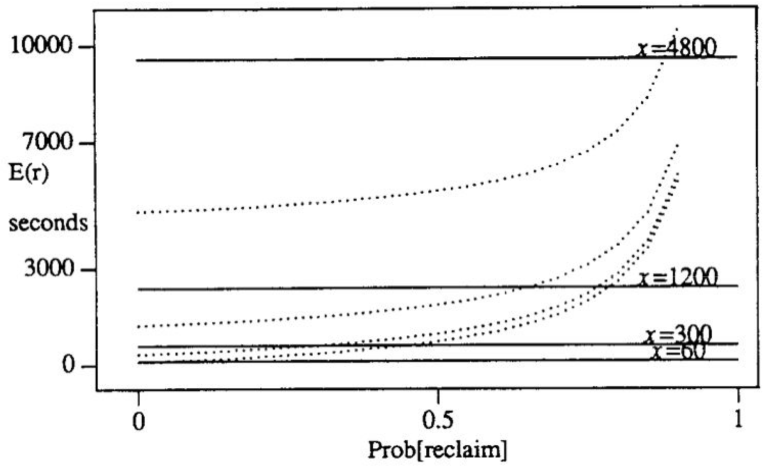


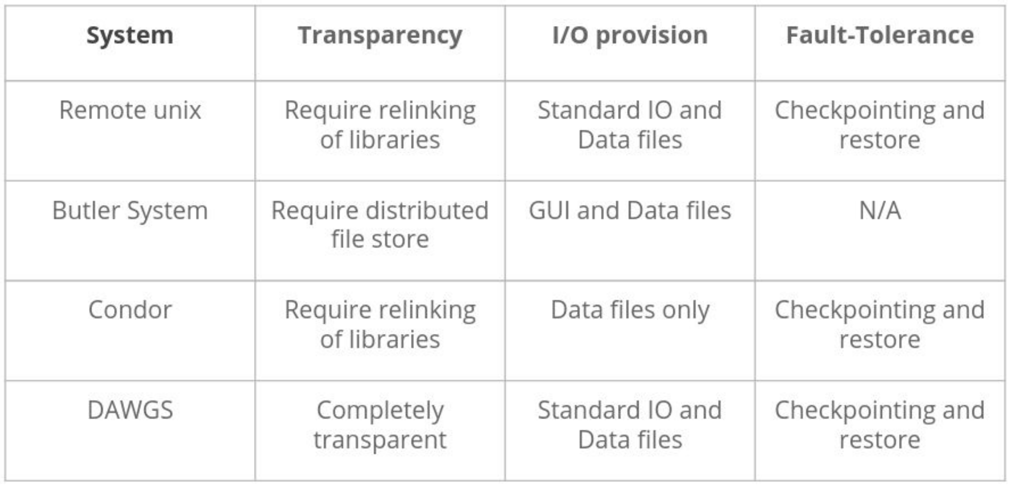
Fig 3 : This graph taken from DAWGS paper represents expected runtime vs probability of reclaim[failure]. Solid line represents local execution runtime and dotted the remote execution runtime. Remote execution time increases exponentially as failure increases.

This method has disadvantages too. The checkpoint method was able to guarantee eventual completion in all cases. But, the active replication method might fail to complete the process if all clones get killed. The probability of process loss is minimized by making more clones. More clones mean occupying more nodes for a process. This blocks more workstations and much time for the process will be spent on waiting to get an idle workstation to execute.

Hence, none could be held one over another. Active replication favors interactive and short compute-bound jobs. Whereas, the checkpoint method is ideal for large compute bound jobs. A hybrid that ensures faster execution and eventual completion guarantee can be proposed as well. We end it here since the scope of this review is limited.

## Conclusion

Remote execution utilizes otherwise wasted computing cycles and provides faster execution than local execution. The syscall redirection mechanism makes it much easier to integrate and transparent as well. This transparent design makes it less efficient compared to its alternatives. Former compromises efficiency for ease of use. Softwares like MPI[8] and OpenMP[9] position in other-end of the spectrum. The mentioned softwares is used widely for simulation and scientific calculations. We have compared the content of four papers regarding transparent remote execution and provided a new fault-tolerant mechanism.



| System | Transparency | I/O provision | Fault-Tolerance |
|---|---|---|---|
| Remote unix | Require relinking of libraries | Standard IO and Data files | Checkpointing and restore |
| Butler System | Require distributed file store | GUI and Data files | N/A |
| Condor | Require relinking of libraries | Data files only | Checkpointing and restore |
| DAWGS | Completely transparent | Standard IO and Data files | Checkpointing and restore |

Fig 4: Summarization table

Table in fig4 summarizes the characteristics of each paper under the Remote environment and fault-tolerance. Support for interactive I/O and data files by DAWGS and the Butler system makes them ideal for interactive process and compute-bound jobs too. Whereas, Condor and Remote Unix support to data files IO make them best suited for compute-bound execution. A fault-tolerant mechanism is required to accommodate the failures of remote processes. Conventional checkpoint method is used by all papers except the butler system. Alternative to checkpoint in terms of efficiency in storage, networks and time is active replication. Yet, Hybrid of the two can provide more useful results.

## Reference

1. M. W. Mutka and M. Livny, “Profiling Workstations’ Available Capacity for Remote Execution” Performance ‘87, Proceedings of the 12th IFIP WG 7.3 Symposium on Computer Performance, Brussels, Belgium, (December 7-9, 1987)

2. M. Litzkow, “Remote Unix”, Proceedings of 1987 Summer Usenix Conferences, Phoenix, Arizona, (June, 1987).

3. D. Nichols. 1987. Using idle workstations in a shared computing environment. In *Proceedings of the eleventh ACM Symposium on Operating systems principles* (*SOSP '87*). Association for Computing Machinery, New York, NY, USA, 5–12.

4. Litzkow, M. J. Livny, M., and Mutka, M. W. Condor-A hunter of idle workstations. Proc. I988 Conference on Distributed Computing Systems, pp. 104-l 11.

5. H. Clark and B. McMillin, "DAWGS - A Distributed Compute Server Utilizing Idle Workstations," *Proceedings of the Fifth Distributed Memory Computing Conference, 1990.*, Charleston, SC, USA, 1990, pp. 732-741.

6. Howard, J.H.; Kazar, M.L.; Nichols, S.G.; Nichols, D.A.; Satyanarayanan, M.; Sidebotham, R.N. & West, M.J. (February 1988). "Scale and Performance in a Distributed File System". *ACM Transactions on Computer Systems*. 6 (1): 51–81.

7. Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, 21(7), July 1978.558-565

8. Walker DW (August 1992). Standards for message-passing in a distributed memory environment. Oak Ridge National Lab., TN (United States), Center for Research on Parallel Computing (CRPC). p. 25.

9. https://www.openmp.org/ [OpenMP]