Documentation:

1. How to build Condor:

1. Get Condor src, untar it. Inside the condor src directory, run cmake using the following command
   cmake -D_DEBUG:BOOL=TRUE -DCMAKE_INSTALL_PREFIX:PATH=`pwd`/install -DBUILDID:STRING=xx
2. If you receive an error mesasge saying libxml2 not found- then install libxml2 by
   sudo apt-get install libxml2
3. If you cant find libxml2 in /usr/lib then export the apporiate path in LD_LIBRARY_PATH
   use *dpkg-query -L <<name>>* to find the path of the installed package
   If it doesnt work, then take a fresh copy of the condor and cmake it(use command specified in point no 1)
4. If the error says uuid, then install uuid-dev
   sudo apt-get install uuid-dev
   Cmake it again(use command in point 1)
5. If it cant find the boost packages then install that
   sudo apt-get install liboost-python-dev
   sudo apt-get install libboost-thread-dev
6. make
7. If make gives error such as dlopen: undefined reference, then run the following commands:
   1. *cd src/condor_tools/*
   2. */usr/bin/c++ -Wall -W -Wextra -Wfloat-equal -Wendif-labels -Wpointer-arith -Wcast-qual -Wcast-align -Wvolatile-register-var -fstack-protector -rdynamic -g -Wl,--warn-once -Wl,--warn-common -ldl -pthread CMakeFiles/condor_gpu_discovery.dir/condor_gpu_discovery.cpp.o CMakeFiles/condor_gpu_discovery.dir/__/condor_utils/condor_version.cpp.o -ldl -o condor_gpu_discovery -rdynamic -Wl,-rpath,"\$ORIGIN/../lib:/lib64:/usr/lib64:\$ORIGIN/../lib/condor"*
   (or)
   /usr/bin/c++ -Wall -W -Wextra -Wfloat-equal -Wendif-labels -Wpointer-arith -Wcast-qual -Wcast-align -Wvolatile-register-var -fstack-protector -rdynamic -g -Wl,--warn-once -Wl,--warn-common -ldl -pthread CMakeFiles/condor_gpu_discovery.dir/condor_gpu_discovery.cpp.o CMakeFiles/condor_gpu_discovery.dir/__/condor_utils/condor_version.cpp.o -ldl -o condor_gpu_discovery -rdynamic -Wl,-rpath,"\$ORIGIN/../lib:/lib64:/usr/lib64:\$ORIGIN/../lib/condor"

8. *make* it again
9. *make install*

Condor should be successfully installed now.

For configuration: (without using ./configure)

1. Changing hostname(solution to the problem found with the mini's)
   vim /etc/hostname
   - Change it to <<ssh_name>>
   vim /etc/hosts

- Check the value corresponding to 127.0.1.1

Modify the hostname

sudo su -

hostname <<ssh_name>>

Check if the hostname is changed using

hostname

2. Find if the <<to_be_created_useraccount>> exists using

egrep <<useraccount> /etc/passwd

If not, create the user account condor using

adduser -u <<number>> <<user_account>>

egrep <<user_account> /etc/passwd

3. Untar the tar file /home.condor.tar.gz in condor directory (it can be found in /p/paradyn/development/salinisk/condor/home.condor.tar.gz)

4. Use chown to change the owner of the files untarred

eg. chown -R condor.condor condor

eg. for specific files chown condor.condor <<path_to_file>>

5. Go to the condor directory i.e. cd condor

Modify condor_config.local as follows

For manager: insert the line

DAEMON_LIST = MASTER, COLLECTOR, NEGOTIATOR

ALLOW_WRITE = host_name of all the machines in the condor pool (eg. for my pool, i will include paradyn-mini@cs.wisc.edu, paradyn-mini2.cs.wisc.edu, paradyn-mini3.cs.wisc.edu

For submit: insert the line

DAEMON_LIST = MASTER, SCHEDD

ALLOW_WRITE = host_name of all the machines in the condor pool (eg. for my pool, i will include paradyn-mini@cs.wisc.edu, paradyn-mini2.cs.wisc.edu, paradyn-mini3.cs.wisc.edu

For execute: insert the line

DAEMON_LIST = MASTER, STARTD

ALLOW_WRITE = host_name of all the machines in the condor pool (eg. for my pool, i will include paradyn-mini@cs.wisc.edu, paradyn-mini2.cs.wisc.edu, paradyn-mini3.cs.wisc.edu

6. Export these environment variables: <<write as a script and source it using the command

. ./<<Script_name>>

Environmental variables

PATH=/home/paradyn/condor-7.9.4/install/bin:$PATH

PATH=/home/paradyn/condor-7.9.4/install/sbin:$PATH

CONDOR_CONFIG=/home/condor/condor_config

7. start/stop condor_master using the command

condor_master

condor_off -master

8. Check if the daemons are started using

```
ps -elf -H | egrep condor
```

In our setup
paradyn-mini: Central manager which runs the negotiator and collector daemon
paradyn-mini2 - submit host, which runs the shedd daemon
paradyn-mini3 - execute host, which runs the startd daemon

To check use:
condor_status - list of all hosts available
condor_staus -master - list of hosts which can connect to your machine
condor_config_val allow_write - will tell us the list of computers who have write access to your comp.
condor_config_val -v allow_write - will tell us where exactly you need to add the line
condor_config_val -config - will tell us the list of configuration files used by the machine.

How to submit jobs to Condor:
Create a file called fibo.c with the following content

```
#include <stdio.h>
#include <stdlib.h>

int fib( int x )
{
	if( x<=0 ) return 0;
	if( x==1 ) return 1;
	return fib(x-1) + fib(x-2);
}

int main(int argc, char *argv[])
{
	int n;
	n = atoi(argv[1]);
	printf ("fib(%d) = %d\n",n,fib(n));
	return 0;
}
```

To compile : gcc –c fib.c –o fib.o
To link: gcc fib.o –o fib
For condor_cmpilation, use the same command, but add condor_compile:
condor_compile gcc fib.o –o fib

How to create a submit file:
fib.submit

```
	Executable = fib
```

Arguments = 4
Output = fib.out
Log = fib.log
queue

Submit the job:
condor_submit fib.submit

Building Self-Propelled Instrumentation(SPI): Refer Self-Propelled Instrumentation documentation.

How to run Condor with SPI:
After Self-propelled Instrumentation is installed successfully and libmyagent is copied from the directory /home/paradyn/spi/user_agent/sc2012_demo/x86_64-unknown-linux2.4/ to the directory /home/paradyn/spi/x86_64-unknown-linux2.4/ ,we can successfully inject SPI into Condor.

SPI can be LD_PRELOAD-ed into Condor by using
LD_PRELOAD=/path/to/libmyagent condor_master
In the paradyn-minis,
LD_PRELOAD=/home/paradyn/spi/x86_64-unknown-linux2.4/libmyagent.so condor_master

Before running this make sure that the following environmental variables are set in paradyn- minis and if you are running in other machines, set the environmental variables to its equivalents

For Condor:
PATH=/home/paradyn/condor-7.9.4/install/bin:$PATH
PATH=/home/paradyn/condor-7.9.4/install/sbin:$PATH
CONDOR_CONFIG=/home/condor/condor_config

For SPI:
export LD_LIBRARY_PATH=/home/paradyn/lib:$LD_LIBRARY_PATH
export DYNINSTAPI_RT_LIB=$HOME/lib/libdyninstAPI_RT.so
export SP_AGENT_DIR=/home/paradyn/spi/x86_64-unknown-linux2.4
export PLATFORM=x86_64-unknown-linux2.4
export LD_LIBRARY_PATH=/home/paradyn/spi/ x86_64-unknown-linux2.4: $LD_LIBRARY_PATH
export SP_NO_TAILCALL=1
export SP_DIR=/home/paradyn/spi
export PATH=$HOME/bin:$PATH

To turn on Self-propelled log, set the environmental variable:
export SP_DEBUG=1

Condor does not like outputs to be sent to the standard output, error or input. redirect it to sepreate files. To turn on Self-propelled log and output to a file, set the environmental variable
export SP_FDEBUG=1

The logs will be outputted to /tmp/spi/ directory in our current implementation of user_agent.

If you are running this on paradyn-minis, the same setup which I used to inject SPI into Condor and run it, use : *sh /home/condor/preload_cond.sh*

How to reproduce the results:
1. LD_PRELOAD SPI into Condor following the instructions from the above section. Wait for all the daemons in all the hosts to start-up.

Central Manager(paradyn-mini)
condor_master, condor_procd, condor_collector, condor_negotiator
Submit_host(paradyn-mini2)
condor_master, condor_procd, condor_schedd
Execute host(paradyn-mini3)
condor_master, condor_procd, condor_startd.

2. Once all the daemons have started and nothing has been killed for atleast 2 mins after its start-up, submit a job in Condor(Follow the section **How to submit jobs in Condor** to submit a job). In our setup, submit a job from account user1 of the submit host(mini2) by running
sh submit_job.sh
3. After the condor_submit has successfully executed, and the result is got, checking the log file of the condor_submit job to figure out how much time it has taken. Sometimes, the normal termination would not happen because of unknown reasons and the job would be put in the idle mode even though the result is obtained.

Some insight about condor_working:
1. Condor_submit contacts condor_schedd and adds job to the job queue
2. condor_schedd sends ClassAd to the condor_collector requesting a machine
3. condor_negotiator matches the request with an available machine
4. condor_schedd claims the machine and spawns a condor_shadow
5. condor_shadow contacts condor_startd and requests the appropriate condor_starter
6. condor_starter actually spawns the application and connects it to the condor_shadow
7. condor_startd monitors the machine and waits for the commands
8. either the application completes or the condor_statrd forces it to either suspend or vacate

Some commands in Condor:
condor_status - list of all hosts available
condor_staus -master - list of hosts which can connect to your machine
condor_config_val allow_write - will tel u the list of computers who have write access to your comp.
condor_config_val -v allow_write - will tel you where exactly you need to add the line
condor_config_val -config - will tell you the list of configuration files used by the machine.

What is not working:

1. As said earlier, sometimes, normal termination of job would not happen.
2. Sometimes, condor_schedd in the submit host would be killed. Not sure what the problem is.
    Round about:
        1. Run Condor without SPI once and remove all the jobs from it.
        2. Run Condor with SPI again.
3. condor_schedd daemon in submit host will be killed invariably after executing every job.

To dos:

1. Current SPI implementation doesnt support UDP
2. Tail calls are not properly handled :

High-level summary of the problem: indirect tail calls come back from ParseAPI (and this should be true for all indirect calls) pointing just to the sink node as a call target. SPI was asserting; what it needs to do is the same thing that Kevin did at unanalysed indirect control flow and instrument before the control transfer to find out where to propagate. Obviously we can improve our static analysis of indirect calls and try to resolve some of these at parse time, but the proper general solution is to intercept before the branch, particularly in the SPI case where we prefer to only instrument paths that are taken in a given run.

3. Try to have SPI daemons running on the machine to avoid many problems.
4. Check the working for 32 bit OS.
5. Do Liveness analysis before storing the registers