LeoTask: a productive and reliable framework for computational research

Changwang Zhang, Shi Zhou, Jie Li and Benjamin M. Chain

This doc is a brief introduction of the LeoTask framework. For more information about LeoTask please refer to:

- Code: `http://github.com/mleoking/leotask`
- Group: `http://groups.google.com/forum/#!forum/leotask`
- Wiki: `http://github.com/mleoking/leotask/wiki`
- Email: changwang.zhang.10@ucl.ac.uk

1 Programming model

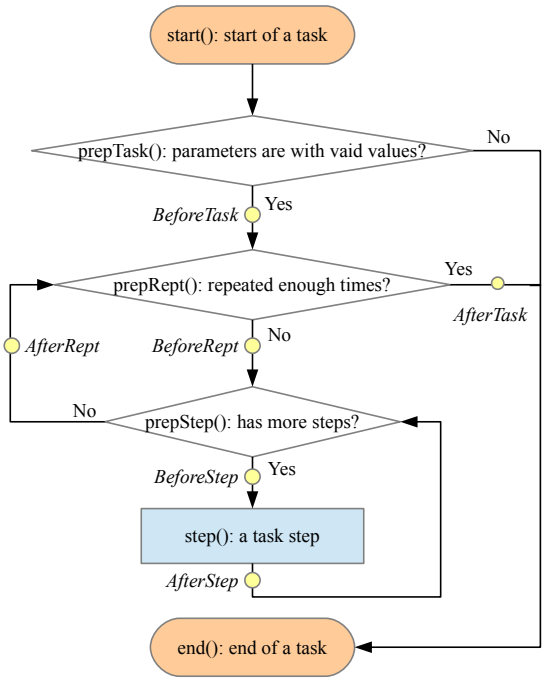

Figure 1: Default method flow and time points for a task. There are 6 default built-in time points for result collection: 1) *beforeTask*: before the start of a task, 2) *beforeRept*: before starting a repeated run of a task, 3) *beforeStep*: before starting a step in a run, 4) *afterStep*: after finishing a step in a run, 5) *afterRept*: after finishing a repeated run of a task, 6) *afterTask*: after finishing a task.

Every application should extend the Task class and implement the application by overriding Task's methods. Figure 1 shows the default method flow for a task. Blow is an example application RollDice. The application simulates to roll a dice at each step. Each dice has *nSide* sides, and there are *nDice* dices. In total it conducts *nDice* steps in each repeat of a task.

```
public class RollDice extends Task {
  public Integer nSide; //Number of dice sides
  public Integer nDice; //Number of dices to roll
  public Integer sum;

  public boolean prepTask() {
    boolean rtn = nSide > 0 && nDice > 0;
    return rtn;
  }

  public void beforeRept() {
    super.beforeRept();
    sum = 0;
  }

  public boolean step() {
```

Table 1: Value expressions.

Expression	Usage	Example
Enumerated values	value1;value2;...	1;2;3;4 = [1, 2, 3, 4]
Repeated values	value@times	2@3 = [2, 2, 2]
Stepped values	from:step:to	1:0.5:3 = [1, 1.5, 2, 2.5, 3]
Nested expressions	Use repeated and stepped expressions within enumerated expressions [1]	0:1:3;5@2;4;8 = [0, 1, 2, 3, 5, 5, 4, 8]
Value blocks	block1}block2}... content in each block will NOT be expanded!	0:1:3}5@2}4}8 = [0:1:3, 5@2, 4, 8]

[1]Evaluation expressions in Table 3 can also be nested in enumerated expressions.

Table 2: Special time points

Name	Description
%plt+	Output the statistic as Gnuplot figure script.
%pltm+	Same as %plt+ but draw multiple variables in one figure when *valVar* includes more than one variable.
%print+	Print the statistic on the screen.
%printm+	Same as %print but print multiple variables together when *valVar* includes more than one variable.

```
    boolean rtn = iStep <= nDice;
    if (rtn) {
      sum += (int) (rand.nextDouble() * nSide);
    }
    return rtn;
  }
}
```

2 Tasks configuration

A configuration file is used to specify the tasks to be conducted. The configuration file is written in the Extensible Markup Language (XML). Below is an example configuration for RollDice:

```
<Tasks>
  <name val="task-rolldice"/><usage val="0.5"/><nRepeats val="5"/><checkInterval val="4"/>
  <variables class="org.leores.task.app.RollDice">
    <nSide val="2;4;6"/>
    <nDice val="2:1:5"/>
  </variables>
  <statistics>
    <members>
      <i><info val="Fig1%pltm+@afterRept@"/>
         <valVar val="sum;#$sum$/$nDice$#"/><parVars val="nSide;nDice"/></i>
      <i><info val="Fig2%plt+@afterRept@"/><valVar val="sum"/><parVars val="nSide"/></i>
      <i><info val="Fig3%plt+@afterRept@"/><valVar val="sum"/><parVars val="nDice"/></i>
    </members>
  </statistics>
</Tasks>
```

The configuration has three sections. The first section includes the values for the parameters of the framework. The example configuration entitles (*name*) the tasks "task-rolldice", allocates (*usage*) 50% CPU cores to run the tasks, sets each task to repeat (*nRepeats*) 5 times, and set checkpoint files (*checkInterval*) to be saved very 4 tasks.

The second section is the *variables* block, setting up the parameters of the task. The values of each task parameter can be configured using the expressions in Table 1. In the example, the task is set to be *org.leores.task.app.RollDice*. The values of RollDice's two

Table 3: Evaluation expressions.

Expression	Usage
Variable	$variable$
Function	$function(parameter list)$
Calculation	#mathematical expression#[1]
Nested	Put variable and functions in calculation. For example: #$v1$+$f1(100)$#

[1]The mathematical expression supports 1) mathematical operators $+, -, *, /, \%, \hat{}$; 2) boolean operators $==, !=, <>, <, <=, >, >=, \&\&, ||$; and 3) functions NOT(exp), RANDOM(), MIN(e1, e2), MAX(e1, e2), ABS(exp), ROUND(exp, precision), LOG(exp), SQRT(exp), SIN(exp), COS(exp), TAN(exp), SINH(exp), COSH(exp), TANH(exp), RAD(exp), DEG(exp).

parameters are provided. *nSide* has 3 values: 2, 4, and 6. *nDice* has 4 values:2, 3, 4, and 5. There are 12 ($= 3 \times 4$) combinations of parameter values in this configuration. 12 tasks each with a different combination of parameter values will be conducted in parallel. Checkpoint files will be saved when the 4^{th}, 8^{th}, and 12^{th} task starts.

The third section is the *statistics* block, describing how and when the task results are aggregated. Each *statistic* (result) has three parameters: *info*, *valVar*, and *parVars*. Parameter *info* includes the time point(s) relevant to the statistic. There are 6 default time points shown in Figure 1 to decide when the statistic is collected. All statistics will be saved in a Comma Separated Values (CSV) file by default. There are some optional special time points, shown in Table 2, to set additional output of the statistics. Each time point ends with "@". Parameter *valVar* and *parVars* together decide how the results are aggregated. For example, a statistic with *valVar*=*y* and *parVars*=*x* analyse *y* conditioned on *x*. *valVar* and *parVars* can include 1) the variables of the task class (e.g. RollDice), and 2) evaluation expressions shown in Table 3.

In the example, there are 3 statistics. They respectively analyse: 1) *sum* and *#sum/$nDice$#* (average outcome of a roll of a dice) conditioned on (*nSide*, *nDice*) pair after each repeat of a task; 2) *sum* conditioned on *nSide* after each repeat of a task; and 3) *sum* conditioned on *nDice* after each repeat of a task.

3 Task running

Tasks can be started either from a program or the command line. Below is an example to run the tasks from a program:

```
public static void main(String[] args) {
 Tasks tasks = new Tasks();
 tasks.sFLoad = "rolldice.xml";
 tasks.start();
 System.exit(0);
}
```

To run the same tasks from the command line:

```
java -jar leotask.jar -load=rolldice.xml
```

The task class (e.g. org.leores.task.app.RollDice) should be either sealed in the leotask.jar or within the class path of Java. Below is the screen output of the example:

```
LeoTask 1.0.0 (C) Changwang Zhang ...
name=rolldice nRepeats=5 seed=1411381224281 ...
cores=4 nThreads=2
Roll Dice 1.0 (C) Changwang Zhang ...
Start Task: 1/12 Elapsed Time: 0.109 Seconds.
Start Task: 2/12 Elapsed Time: 0.109 Seconds.
Start Task: 3/12 Elapsed Time: 0.14 Seconds.
Checkpoint saved to: rolldice_0922@1120_check.4.ckp
Start Task: 4/12 Elapsed Time: 0.172 Seconds.
Start Task: 5/12 Elapsed Time: 0.172 Seconds.
Start Task: 6/12 Elapsed Time: 0.172 Seconds.
Start Task: 7/12 Elapsed Time: 0.172 Seconds.
Checkpoint saved to: rolldice_0922@1120_check.8.ckp
Start Task: 8/12 Elapsed Time: 0.187 Seconds.
Start Task: 9/12 Elapsed Time: 0.187 Seconds.
Start Task: 10/12 Elapsed Time: 0.187 Seconds.
Start Task: 11/12 Elapsed Time: 0.187 Seconds.
Checkpoint saved to: rolldice_0922@1120_check.12.ckp
Start Task: 12/12 Elapsed Time: 0.203 Seconds.
Log statistic data Elapsed Time: 0.018 Minutes.
Finished 12/12 Tasks in 0.01978. Minutes.
```

3.1 Interruption recovery

To recover from interruption, one simply needs to rename the latest checkpoint file to "`check.ckp`" and start running the tasks either from a program or from the command line. The output when recovering from "`rolldice_0922@1120_check.8.ckp`" (i.e. rename it to "`check.ckp`") is shown below:

```
Loading checkpoint: check.ckp ...
Loaded from checkpoint: check.ckp.
LeoTask 1.0.0 (C) Changwang Zhang ...
name=rolldice nRepeats=5 seed=1411381224281 ...
cores=4 nThreads=2
```

```
Roll Dice 1.0 (C) Changwang Zhang ...
*Start Task: 6/12 Elapsed Time: 0.031 Seconds.
*Start Task: 7/12 Elapsed Time: 0.031 Seconds.
Checkpoint saved to: rolldice_0922@1120_check.8.ckp
Start Task: 8/12 Elapsed Time: 0.047 Seconds.
Start Task: 9/12 Elapsed Time: 0.063 Seconds.
Start Task: 10/12 Elapsed Time: 0.063 Seconds.
Start Task: 11/12 Elapsed Time: 0.063 Seconds.
Checkpoint saved to: rolldice_0922@1120_check.12.ckp
Start Task: 12/12 Elapsed Time: 0.078 Seconds.
Log statistic data Elapsed Time: 0.0169 Minutes.
Finished 12/12 Tasks in 0.019 Minutes.
```

There is a "*" for task 6 and 7 indicating that they were running (not finished yet) when the interruption happened and thus are restarted. **Note:** please delete the check point file "`check.ckp`" after finishing the tasks, otherwise LeoTask will always read and recover from that check point.

4 Results files

4.1 Comma Separated Values (CSV) files

The analysed results are by default saved into a plain text based Comma Separated Values (CSV) file. The file presents results in both detailed format and compact format. The detailed format (Table 4 and Table 5) includes all values, and their Max, Min, Average, and Standard Deviation (uncorrected); while the compact format (Table 6) only shows the average values.

Table 4: Detailed statistics for *sum* conditioned on (*nSide*, *nDice*) pair.

nSide	nDice	Max	Min	Std	Avg	N	Vals
2	2	2	0	0.748331	0.8	5	[1,0,1,2,0]
2	3	2	1	0.4	1.2	5	[1,1,2,1,1]
2	4	4	1	1.16619	2.2	5	[1,3,1,2,4]
2	5	5	1	1.356466	2.6	5	[1,3,2,5,2]
4	2	5	0	1.624808	2.4	5	[2,2,3,5,0]
4	3	5	2	1.019804	3.4	5	[3,4,5,3,2]
4	4	10	3	2.712932	5.8	5	[4,8,3,4,10]
4	5	12	4	2.828427	7	5	[4,8,5,12,6]
6	2	9	1	2.785678	4.2	5	[4,2,5,9,1]
6	3	9	4	1.720465	6.2	5	[5,6,9,7,4]
6	4	15	6	3.867816	9.8	5	[6,14,7,7,15]
6	5	18	6	4.214262	11.8	5	[6,15,10,18,10]

The results are collected after each repeat of a task. There are 5 values of *sum* for each pair of (*nSide*,*nDice*); they correspond to the value of *sum* in each repeat of a task.

Table 5: Detailed statistics for *sum* conditioned on *nDice*.

nDice	Max	Min	Std	Avg	N	Vals
2	9	0	2.362673	2.466667	15	[2,2,...,1]
3	9	1	2.360791	3.6	15	[1,1,...,4]
4	15	1	4.186752	5.933333	15	[1,3,...,15]
5	18	1	4.828618	7.133333	15	[1,3,...,10]

The results are collected after each repeat of a task. The *sum* of tasks with a same $nDice$ value but different $nSide$ values are aggregated. There are 3 different $nSide$ values (2,4,6) and each task is repeated 5 times. So that there are in total 15 ($= 3 \times 5$) values of *sum* collected for each $nDice$ value.

4.2 Gnuplot scripts

Gnuplot is a piece of free, open-source, and widely used plotting software. Through special time points (Table 2), results can additionally be represented as figures in Gnuplot scripts. Figure 2 shows the three result figures, each corresponds to a statistic in the example configuration file. These figures are produced by executing the generated (by LeoTask) Gnuplot scripts using Gnuplot 4.6.5.

5 Plot customization

The most straightforward way to customize the plots is to modify the generated Gnuplot script files. For the language of Gnuplot, please refer to its website: `http://www.gnuplot.info/`.

The plots can also be customized through the built-in hybrid programming engine for Java and Gnuplot. The benefits of this approach is that the result figures will be in the customized style directly after finishing all tasks; there is no need to manually modify the

Table 6: Statistic results in the compact format showing only the average.

nSide	nDice	sum	#sum/$nDice$#
2	2	0.8	0.4
2	3	1.2	0.4
2	4	2.2	0.55
2	5	2.6	0.52
4	2	2.4	1.2
4	3	3.4	1.133333
4	4	5.8	1.45
4	5	7	1.4
6	2	4.2	2.1
6	3	6.2	2.066667
6	4	9.8	2.45
6	5	11.8	2.36

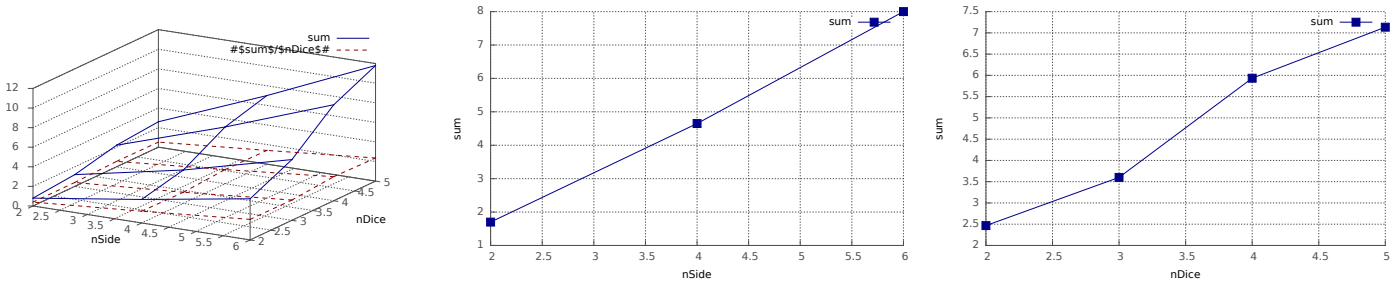

Figure 2: Result figures for Roll Dice. Each figure corresponds to a statistic in the configuration file.

Gnuplot scripts after every run of the tasks. For more details about the hybrid programming engine, please refer to the demo class: *org.leores.demo.JGnuplotDemo*.

A task class can override the afterAll function in the Task class to customize the plots. Blow is an example. The customized result figures are shown in Figure 3. **Note:** the following code requires Gnuplot installed and its *gnuplot* command directory included in the system's PATH environment variable.

```
public static void afterAll(Tasks tTasks) {
 Task.afterAll(tTasks);
 Statistics stats = tTasks.getStatistics();
 //Get the data results
 DataTableSet dts1 = stats.getDataTableSet(null, "Fig1.*");
 DataTableSet dts2 = stats.getDataTableSet(null, "Fig2.*");
 DataTableSet dts3 = stats.getDataTableSet(null, "Fig3.*");

 JGnuplot jg = new JGnuplot() {
   {
    terminal = "pdfcairo enhanced dashed size 5,3";//set to output pdf
    output = "$info$.pdf";
    beforeStyleVar = "lw=4;";//set the line width to 4
    extra = "unset grid;";
   }
 };

 Plot plot1 = new Plot("fig1x") {
   {
    xlabel = "No. of sides";
    ylabel = "No. of dices";
    extra2 = "set key at screen 0.9,0.9,0.9;";
   }
 };
 plot1.add(dts1);
 dts1.get(0).info = "Sum";
 dts1.get(1).info = "Sum / No. of dices";
 jg.execute(plot1, jg.plot3d);

 Plot plot2 = new Plot("fig2x") {
```

```
  {
    xlabel = "No. of sides";
    ylabel = "Sum";
    yrange = "[0:9]";
    extra2 = "unset key;";
  }
 };
 plot2.add(dts2);
 String sBarplot = "$style2d$\n$header$\n";
 sBarplot += "plot '-' using 2:xtic(1) w histograms;\n" + "$data(1,2d)$\n";
 jg.execute(plot2, sBarplot);

 Plot plot3 = new Plot("fig3x") {
  {
    xlabel = "No. of dices";
    ylabel = "Sum";
    extra2 = "unset key;";
  }
 };
 plot3.add(dts3);
 jg.execute(plot3, jg.plot2d);
}
```

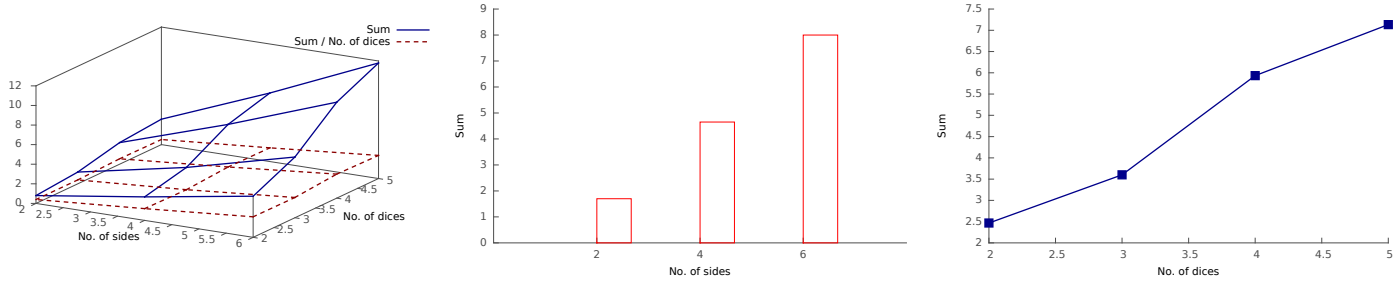

Figure 3: Same result as in Figure 2 but in the customized plotting style.