Репозиторий файлов для проведения практических занятий

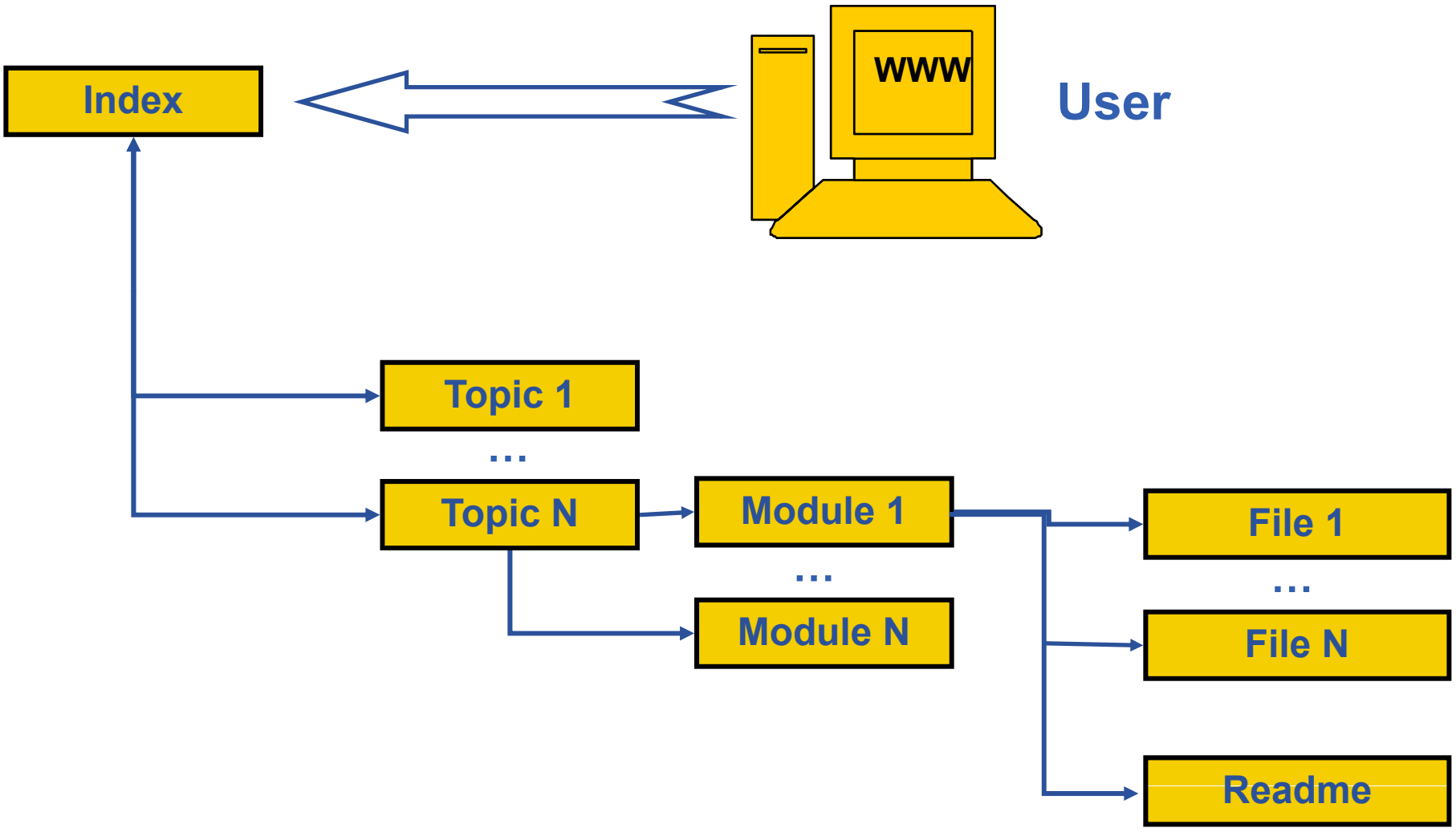
Клопов Николай, Олешко Сергей
Петербургский институт ядерной физики РАН, Гатчина.

Проблемы и цель создания

- Практические задания (скрипты, программы, и т.п.) доступны только со страниц курсов
- После проведения курсов они могут быть удалены
- В Digital Library, как правило, либо описания, либо презентации с включёнными текстами

Цель – создание и поддержка единого репозитория файлов для практикумов с удобным доступом.

Структура


Index
WWW
User
Topic 1
...
Topic N
Module 1
...
Module N
File 1
...
File N
Readme

Темы

- **Работа с утилитами информационной системы**
- Запуск заданий
- Работа с данными
- Работа в BO LCG
- ……???

Работа с утилитами ИС

- Набор команд lcg-infosites, lcg-info
- Набор команд ldapsearch

Темы

- Работа с утилитами информационной системы
- **Запуск заданий**
- Работа с данными
- Работа в BO LCG
- ……???

Запуск заданий в ГРИД

- Простое задание
- Interactive job
- DAG
- Parametric job
- Job Collection
- MPI

Простое задание 1

- **Цель: Запуск готовой программы с входными параметрами и входными данными. Получение файла результатов.**
- **Требуемые файлы:**
  - программа на PERL tstp.pl, которая использует входной файл с колонкой чисел. Каждое число возводится в квадрат и записывается в выходной файл. Имя входного файла передается как параметр программы. Имя выходного файла то же как и входного, но с расширением ‘out’.
  - входной файл с числами.
  - JDL файл.

Простое задание 1

- JDL файл:

```
Executable = «tstp.pl";
Arguments = "pinp.inp";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox={“tstp.pl","pinp.inp"};
OutputSandbox = {"std.out", "std.err","pinp.out"};
```

Простое задание 2

- **Цель: Компиляция и сборка программы на WN.**
- **Требуемые файлы:**
  - исходный файл программы на C, выводит “Hello world” на стандартный вывод
  - JDL файл.
  - Makefile для компиляции и сборки программы.
  - выполняемый bash скрипт.

Пример задания 2

- JDL файл:

```
JobType="Normal";
Executable = "startC.sh";
StdOutput = "ctst.out";
StdError = "ctst.err";
OutputSandbox = {"ctst.out","ctst.err"};
InputSandbox = {"startC.sh","ctst.c","Makefile"};
```

Пример задания 2

- Стартовый скрипт startC.sh:

```
#!/bin/bash
make                    //сборка программы
// chmod +x ctst   //разрешаем запускать ее (
./ctst                  //запускаем
exit 0
```

Пример задания 3

- Есть файл со скриптом, который генерит набор JDL файлов, количество которых определяется входным параметром
- Затем для каждого задания случайным образом выбираются слова из системного словаря /usr/share/dict/words, которые передаются, как аргументы для каждого из запускаемых заданий.
- Все задания запускаются и контролируется процесс их выполнения.
- После завершения всех заданий (успешного или нет) – выводится результат.
- Требуемые файлы:
  - файл со скриптом
  - исполняемое задание для вывода слова

Пример задания 3

- Файл echoword.sh

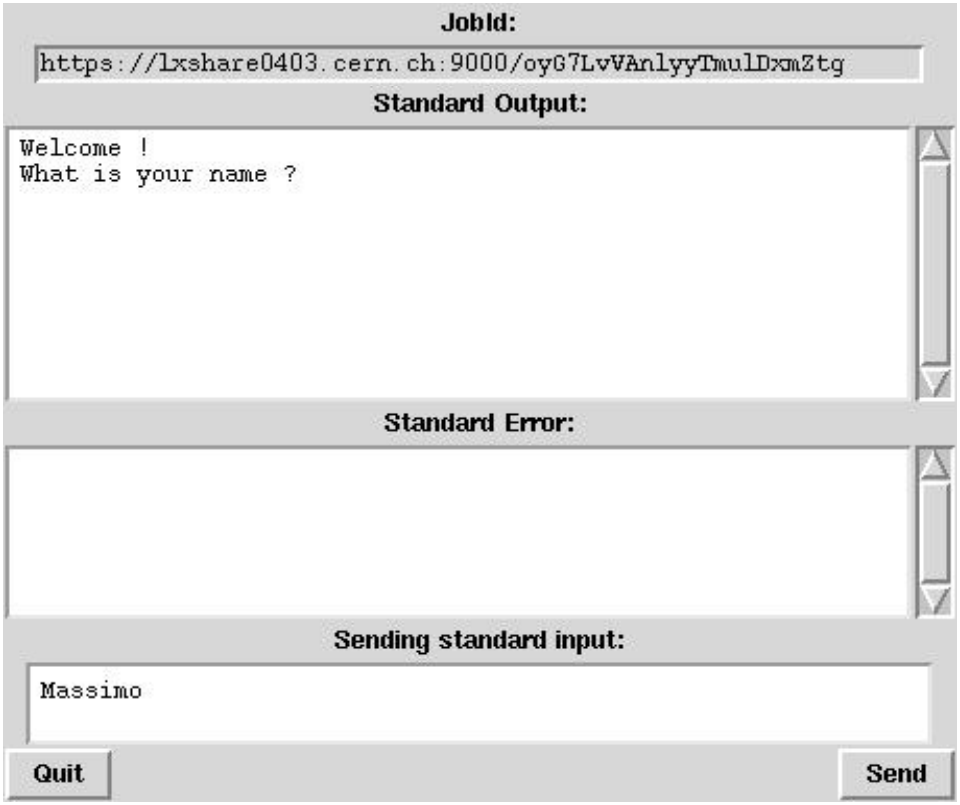
```
#!/bin/bash
echo "Word $1 is $2";
```

- Запуск задания

./submit-dictionary-jobs.sh 3

Interactive job

- Специфицируется установкой JobType = “Interactive” в JDL
- Когда interactive job запускается, открывается окно для stdin, stdout, stderr потоков

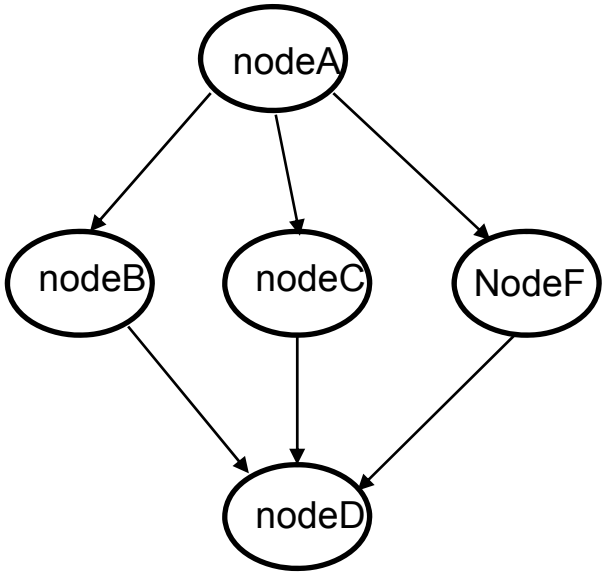


Interactive job

- **Цель: Демонстрация возможностей интерактивных заданий в Грид.**
- **Требуемые файлы:**
  - Executable bash-скрипт, принимающий сообщения от пользователя и выдающий приглашение на следующее сообщение.
  - JDL файл.

DAG job

- DAG (Directed Acyclic Graph)– это набор задач, где ввод, вывод или выполнение одних задач может зависить от других
- Зависимости представляются графом, где нодами являются задачи, а ребра определяют зависимости
- Зависимости не могут быть циклическими (Acyclic Graph)
- DAG в целом (как и его ноды) получает уникальный ID, который может использоваться для управления (cancel, get status, get output)



DAG job

```
[
  type = "dag";
  VirtualOrganisation = "ATLAS";
  max_running_nodes = 4;
 Requirements=other.GlueCEInfoTotalCPUs>2;
  InputSandbox = ...
 nodes = [
    nodeA = [
      file ="nodes/nodeA.jdl" ;
    ];
    nodeB = [
      file ="nodes/nodeB.jdl" ;
    ];
    nodeC = [
      file ="nodes/nodeC.jdl" ;
    ];
    nodeD = [file ="nodes/nodeD.jdl";];
  dependencies = {
      {nodeA, nodeB},{nodeA, nodeC},
      { {nodeB,nodeC}, nodeD }
    };
 ];
]
```

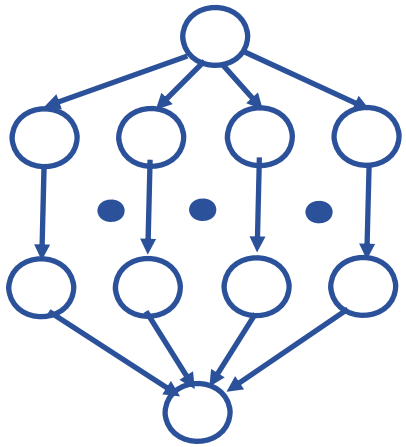
Набор JDL атрибутов, описывающих DAG в целом и которые могут наследоваться нодами (root section)

Node section

Зависимости

DAG

- Цель: Демонстрация возможностей заданий типа DAG в Грид.
- Для генерации JDL файла, определяющего DAG произвольной структуры, были разработаны соответствующие модули на Perl и Python: DAG.pm, DAG.py
- При инициации модуль генерирует регулярную JDL DAG структуру(N layers, M nodes):



- Layer 1
- Layer N

DAG

Модуль DAG обеспечивает следующие возможности:

```
#              Class for DAG generation
#  createDAG- DAG creation by template
#  clear- clearning object
#  addDeps- add new dependencies to DAG
#  removeDeps- remove some dependencies from DAG
#  addNodes- add new nodes with JDL attributes to DAG
#  removeNodes- remove some nodes from  DAG
#  replaceNodesJdl- replace jdl attributes for given nodes
#  addRootOption- add new attribute into root section of DAG
#  removeRootOption- remove given attribute from DAG
#  addOptions- add list of attributes to given nodes
#  removeOption- remove given attribute from given list of nodes
#  addRequirement- add requirement (with logical AND) to given list of
   nodes
#  addRootRequirement- add requirement (with logical AND) to root section
#  saveToFile- save DAG-jdl into given file and nodes if need into
   corresponding
```

Parametric job

- **Parametric job- это задание, где один или более ее атрибутов параметризируемы и для каждого значения параметра создается отдельная задача**

```
[
  JobType = "Parametric";
  Executable = "/bin/sh";
  Arguments = "md5.sh input_PARAM_.txt";
  InputSandbox = {"md5.sh", "input_PARAM_.txt"};
  StdOutput = "out_PARAM_.txt";
  StdError = "err_PARAM_.txt";
  Parameters = 10000;
  ParameterStart = 1000;
  ParameterStep = 100;
 OutputSandbox={"out_PARAM_.txt","err_PARAM_.txt"};
]
```

Запуск такого JDL приведет к генерации N задач, где

N = (Parameters – ParameterStart)/ParameterStep

- **Мониторирование и управление заданием производится как через уникальный ID всего задания, так и через назначенные ID для отдельных задач**

Parametric job

Для демонстрации параметрических заданий можно использовать программу prmtest.pl, которая разрабатывалась для тестирования. Эта программа может просто сгенерировать требуемый JDL файл для параметрического задания и сохранить его на диске.

Например,

```
./prmtest.pl –n 100 –j
```

создаст JDL файл параметрического задания, где атрибут Parameters=100. Опция –j требует завершить работу программы после генерации JDL.

Job Collection

- Job collection- это набор независимых задач, который пользователь может запустить и мониторировать одним запросом
- Задачи из Job collection запускаются как DAG ноды без зависимостей (dependencies)
- JDL для описания Job collection:

```
[
        Type = "collection";
.....................................................
        nodes = {
                [  <job descr 1 >],
                [  <job descr 2 >],
                 …
               };
]
```

Пример Job Collection

```
[
  type = "collection";
  InputSandbox = {"date.sh"};
  RetryCount = 0;
  nodes = {
     [
       file ="jobs/job1.jdl" ;
     ],
     [
       [  NodeName = “job2”;
          Executable = "/bin/sh";
          Arguments = "date.sh";
          Stdoutput = "date.out";
          StdError =  "date.err";
          OutputSandbox ={"date.out", "date.err"};
        ]
     ],
     [
       file ="jobs/job3.jdl" ;
     ]
  };
]
```

Все узлы разделяют использование этого Input Sandbox

Job Collection

Набор независимых задач может быть запущен в gLite с использованием нового параметра –collection в команде

glite-wms-job-submit:

glite-wms-job-submit –collection <dirpath> …….

Эта опция позволяет определить путь до директории, которая содержит JDL файлы независимых задач, составляющих требуемую коллекцию.

Команда glite-wms-job-submit сама создаст соответствующий JDL с типом Type=collection и запустит задание в gLite.

Файлы для тренинга в настоящее время отсутствуют

MPI

- **Возможность выполнения параллельных задач является важным требованием к GRID инфраструктуре**
- **Наиболее используемая библиотека для поддержки параллельных задач является MPI** (Message Passing Interface)
- **В настоящее время параллельные задачи могут выполняться *только* на одном Computing Elements (CE)**
- **Задание должно быть скомпилировано с библиотеками mpicc**

MPI job

- **С точки зрения клиента, задачи, которые должны выполняться как MPI, специфицируются установкой JDL JobType атрибута в MPICH, а также NodeNumber атрибута.**

Например:

```
…
JobType = “MPICH”;
NodeNumber = 4;
…
```

Этот атрибут определяет требуемое число ЦПУ, необходимых приложению

MPI job

```
[
 Type = "Job";
#Обязательный параметр
 JobType = "MPICH";
 Executable = "cpi";
#Количество ЦПУ, которое будет использовано
 NodeNumber = 2;
 StdOutput = "cpi.out";
 StdError = "cpi.err";
 InputSandbox = {"cpi"};
 OutputSandbox = {"cpi.err","cpi.out"};
 RetryCount = 0;
]
```

Темы

- Работа с утилитами информационной системы
- Запуск заданий
- **Работа с данными**
- Работа в BO LCG
- ……???

Работа с данными

- Интерактивный сеанс с созданием, сохранением и регистрацией файла в каталоге
- Варианты простой работы с данными из скрипта (копирование, сохранение)
- Использование GFAL API

Темы

- Работа с утилитами информационной системы
- Запуск заданий
- Работа с данными
- **Работа в BO LCG**
- ……???

Работа в BO LCG

Темы

- Работа с утилитами информационной системы
- Запуск заданий
- Работа с данными
- Работа в BO LCG
- ……???

Заключение

- Создание репозитория входит в планы по EGEE-III
- Предполагаемое размещение – на сайте ПИЯФ (http://egee.pnpi.nw.ru)
- Ответственность за поддержку репозитория – ПИЯФ
- NA3 РДИГ партнёры (ОИЯИ, ИФВЭ) участвуют в его пополнении