# Workflows Científicos

Diego Carvalho

Autores: Diego Carvalho, Luiz Gadelha, Kary Ocaña
Programa de Verão do LNCC, 2021
Laboratório Nacional de Computação Científica

14 de janeiro de 2021

Laboratório
Nacional de
Computação
Científica

# Agenda MC-SD05-I Workflows Científicos

Introdução

Experimentos Científicos

Workflows Científicos

Parsl

Mão na massa

Conclusão

Laboratório Nacional de Computação Científica

# Table of Contents


Introdução

Experimentos Científicos

Workflows Científicos

Parsl

Mão na massa

Conclusão


Laboratório Nacional de Computação Científica

# Introdução

- Experimentos Científicos Computacionais
- Jim Gray disse que e-Science é onde a "tecnologia da informação se junta aos cientistas"
- Uma parte crescente da ciência e tecnologia se apoia em experimentos computacionais.
- Características destes experimentos:
  - grande quantidade de tarefas computacionais;
  - grande quantidade de dados;
  - dados armazenados em formatos diversos;
  - utilização de computação paralela e distribuída.

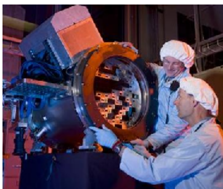
- Experimentos científicos computacionais (*in silico*) atingiram uma escala na qual:
  - é difícil tratá-los com recursos de apenas uma instituição;
  - é difícil gerenciá-los "manualmente" (scripts);
  - é difícil analisar os seus resultados, em função da quantidade de dados gerada.
- Estes experimentos são intensivos em termos de **processamento** e em termos de **dados**.

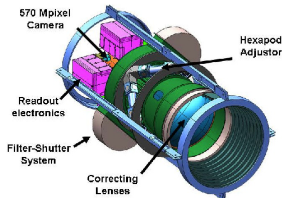
- Exemplos:
  - Modelos climáticos mais recentes geram 8TB para uma simulação de 100 anos com resolução de 100km, e 8PB para resolução de 3km.
  - *Large Hadron Collider* gera 10PB de dados brutos por dia 50PB de dados tratados por ano.
  - O SKA (*Square Kilometre Array*) será composto por 3.000 rádio-telescópios de 15m de diâmetro. Envolve 70 instituições de 20 países. O volume de dados brutos gerado diariamente será de 1 Exabyte, (2x o que trafega pela Internet por dia atualmente). Volume pós-processamento: 300 Petabytes a 1,5 Exabyte por ano.

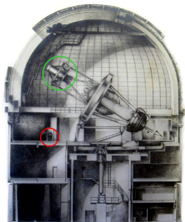
R. Kouzes et al. The Changing Paradigm of Data-Intensive Computing. *IEEE Computer*, 42(1):26-34, 2009.

LHC the guide, `https://press.cern/press-kit`, 2017.

# Exemplo: Dark Energy Survey (DES)



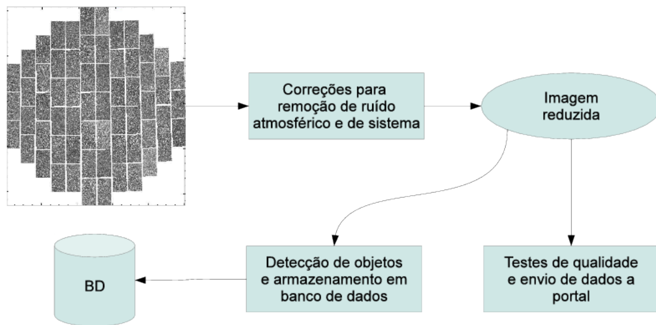


- Câmera de 570 megapixels.
- 400 imagens (6,6TB) por noite.

Dark Energy Survey (http://www.darkenergysurvey.org)

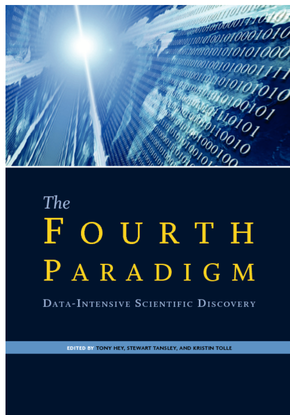
# Exemplo: Workflow do DES




K. Kotwani et al. The Dark Energy Survey Data Management System as a Data Intensive Science Gateway. *Proc. of the 8th International Workshop on Middleware for Grids, Clouds and e-Science* (MGC 2010). ACM, 2010.

Laboratório Nacional de Computação Científica

# Quarto Paradigma Científico



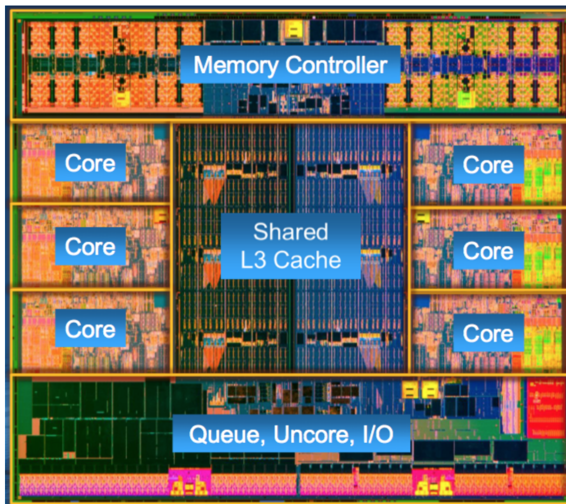

- Paradigmas anteriores: empírico, teórico e computacional (simulação).
- Habilitado por instrumentos científicos e simulações que geram grandes massas de dados.
- O quarto paradigma é baseado na análise e exploração destas massas de dados. Também chamado de e-Ciência.

http://research.microsoft.com/en-us/collaboration/fourthparadigm/

Laboratório Nacional de Computação Científica

Processador Apple A8 (iPhone 6) com 2 núcleos. Fonte: Chipworks.

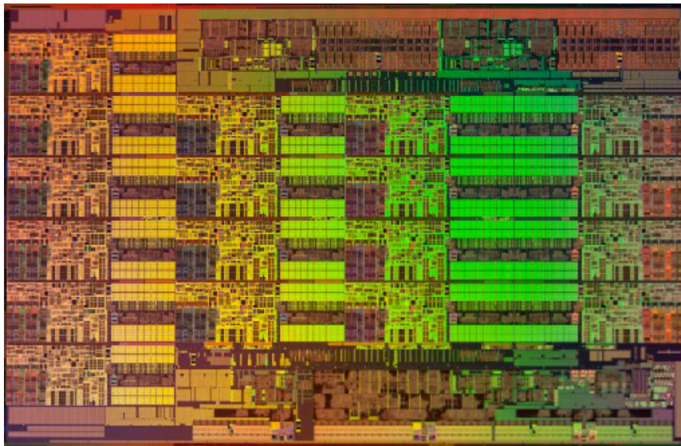
# Recursos Computacionais




Processador Intel i7-4960X com 6 núcleos de processamento. Fonte: Intel.

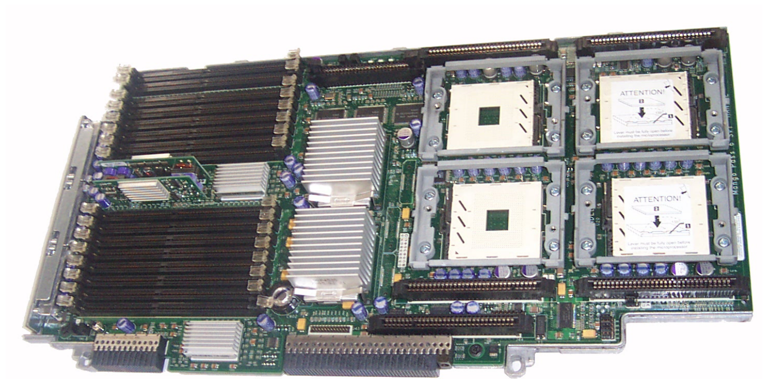
Laboratório Nacional de Computação Científica

# Recursos Computacionais



Processador Intel Xeon E7-8895 v3 com 18 núcleos de processamento. Fonte: Intel.

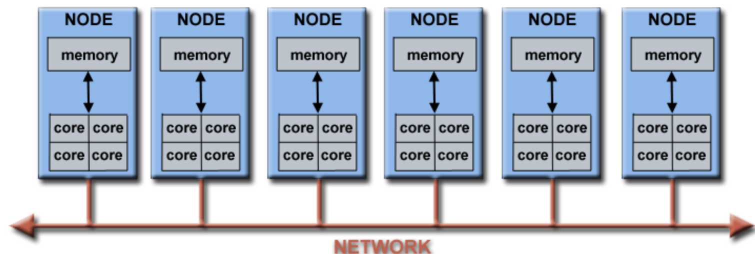
Laboratório Nacional de Computação Científica




IBM EServer xSeries 445 SMP Board. Fonte: IBM.

Laboratório Nacional de Computação Científica

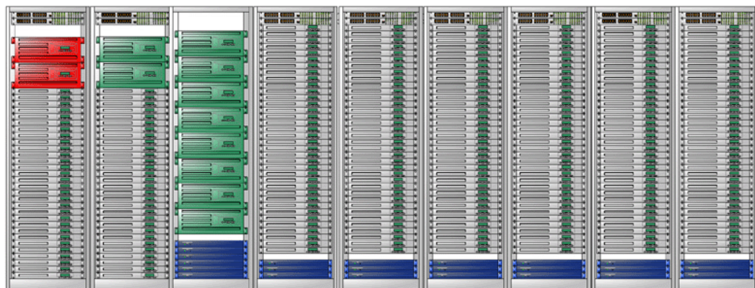
- *Cluster* de computadores:

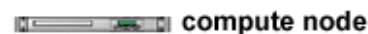



Fonte: Blaise Barney. Introduction to Parallel Computing. Lawrence Livermore National Laboratory. https://computing.llnl.gov/tutorials/parallel_comp/

# Recursos Computacionais

- *Cluster* de computadores:



 compute node

login / remote partition server node

infiniband switch

gateway node

management hardware

Fonte: Blaise Barney. Introduction to Parallel Computing. Lawrence Livermore National Laboratory. `https://computing.llnl.gov/tutorials/parallel_comp/`

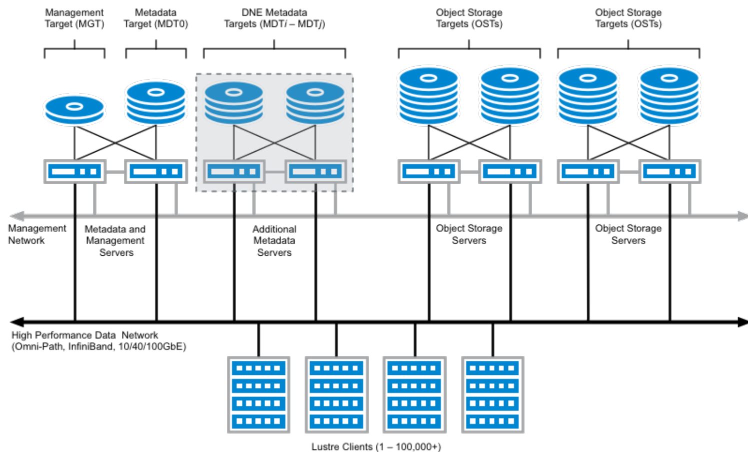
# Recursos Computacionais

Hierarquia

- núcleo
- processador
- nó de processamento
- cluster
- site
- centro de recursos

# Recursos Computacionais

- *Lustre* sistema de arquivamento:




Fonte: Lustre Wiki. `https://wiki.lustre.org/Introduction_to_Lustre`

Laboratório Nacional de Computação Científica

# Table of Contents

Introdução

Experimentos Científicos

Workflows Científicos

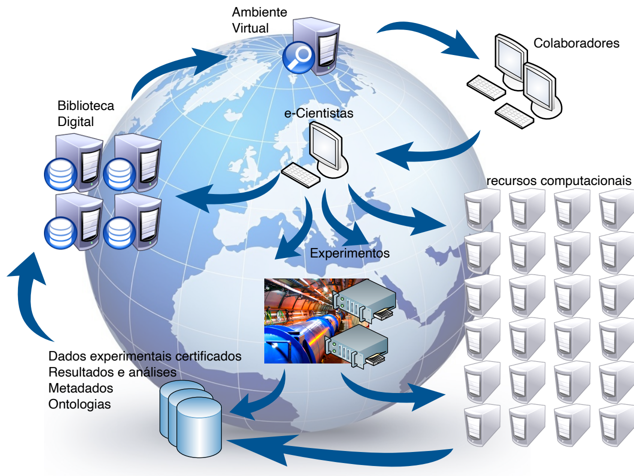
Parsl

Mão na massa

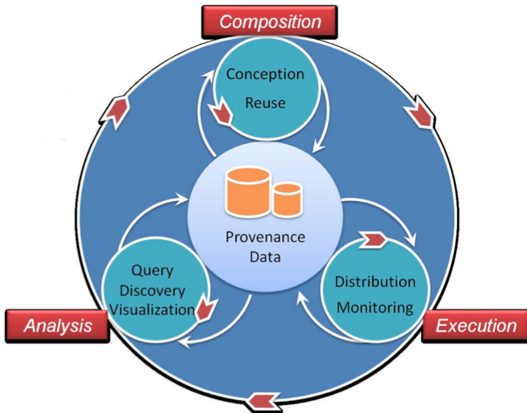
Conclusão

Laboratório Nacional de Computação Científica

# Experimentos Científicos Computacionais



# Ciclo de Vida de Experimentos Científicos




M. Mattoso, et al., Towards supporting the life cycle of large scale scientific experiments.

*International Journal of Business Process Integration and Management* 5(1):79–92, 2010.

Laboratório Nacional de Computação Científica

- **Composição**. Especificação das atividades e dos dados que serão utilizados. Pode incluir o reuso de especificações de experimentos similares.
- **Execução**. Mapeamento, distribuição e inicialização das atividades do experimento em tarefas concretas executadas em recursos computacionais. Inclui também o monitoramento das mesmas e a movimentação dos dados necessários.
- **Análise**. Avaliação do experimento através dos resultados obtidos, p. ex. dados de saída. Pode se basear também em consultas aos registros de eventos da execução (proveniência).

M. Mattoso et al. Desafios No Apoio à Composição De Experimentos Científicos Em Larga Escala. *Anais do XXXVI Seminário Integrado De Software e Hardware* (SEMISH), *XXIX Congresso Da Sociedade Brasileira De Computação* (CSBC), p. 307-321, 2009.

# Requisitos de Experimentos Científicos Computacionais

- **Abstração**. Deve ser possível ao cientista especificar o experimento em alto nível, sem precisar detalhar aplicações específicas e onde elas serão executadas.
- **Reprodutibilidade**. O experimento pode ser reproduzido por terceiros de forma independente.

  D. Koop et al. A Provenance-Based Infrastructure for Creating Executable Papers. *Proceedings of the ICCS*, 2011.
- **Reutilização**. Deve ser possível reutilizar especificações de experimentos realizados previamente para concepção de novos experimentos.

  Exemplo: MyExperiment (`http://www.myexperiment.org`).
- **Escalabilidade**. O experimento deve ser resiliente ao aumento do número de tarefas e da quantidade de dados.

Laboratório Nacional de Computação Científica

# Experimentos Científicos Computacionais em Larga Escala

- Experimentos em larga escala normalmente requerem o uso de computação paralela e distribuída:
  - **Computação paralela**. Utilização de múltiplos processadores de forma concorrente para reduzir o tempo de execução de atividades.

    I. Foster. Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. Addison-Wesley, 1994.
  - **Computação em grade**. Compartilhamento de recursos computacionais heterogêneos e distribuídos.

    I. Foster e C. Kesselman, Eds. The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 2003.
  - **Computacão na nuvem**. Utilização de recursos de terceiros.

    D. Oliveira, F. Baião e M. Mattoso. Towards a Taxonomy for Cloud Computing from an e-Science Perspective. Cloud Computing, pp. 47–62. Springer, 2010.

Laboratório Nacional de Computação Científica

- Os grandes experimentos utilizam de paralelismo em diversos níveis, variando de clusters até HPC
- Diversas ferramentas computacionais dos experimentos exploram códigos paralelos para viabilizar experimentos

# Grades Computacionais

- A realização destes experimentos requer o compartilhamento de recursos computacionais distribuídos.
- Computação em grade permite ganhos de escala em termos de processamento.
- Gerência de dados em grades é uma área em desenvolvimento.

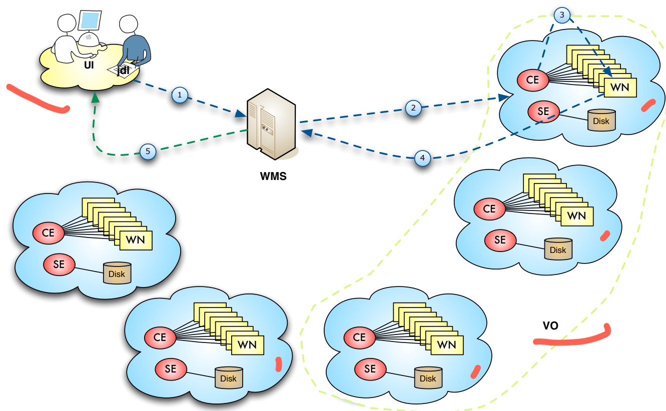
Foster, I. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 15(3), 200–222.
http://doi.org/10.1177/109434200101500302

Laboratório Nacional de Computação Científica

# Grades Computacionais

- Uma **grade computacional** é uma infra-estrutura composta de software e hardware que provê recursos computacionais de alto desempenho.
- Tais recursos podem estar distribuídos entre múltiplas instituições sob domínios administrativos distintos.
- O *Globus* é um conjunto de ferramentas para implementação de grades computacionais.

- Malha:




Fonte: D.Carvalho. Controle distribuído de Workflows em malhas computacionais. Rio de Janeiro: UFRJ/COPPE, 2010.

# Exemplo: Large Hadron Collider (LHC)

- Acelerador de partículas de 27km de circunferência localizado na fronteira da França e Suíça (CERN).
- Projeto colaborativo envolvendo cerca de 100 países e 10.000 cientistas.
- Composto por seis detectores que geram cerca de 15PB de dados por ano.
- LHC Computing Grid (LCG):
  - Estrutura de computação distribuída organizada de forma hierárquica, para execução de diferentes partes do workflow do experimento:
    - Tier 0: processamento de dados brutos dos detectores,
    - Tier 1: reconstrução e pré-processamento,
    - Tier 2: simulação e análise.

Laboratório Nacional de Computação Científica

# Exemplo: Mapa do LCG

Fonte: CERN (`https://wlcg-public.web.cern.ch/`)

Laboratório Nacional de Computação Científica

# Computação na Nuvem

- Consiste das aplicações disponibilizadas como serviços através da Internet e da infraestrutura (hardware e software) necessária para disponibilizá-las.
- Diferentes tipos de serviços:
  - *Software as a Service* (SaaS): Google Docs.
  - *Platform as a Service* (PaaS): Google App Engine, Amazon Lambda.
  - *Infrastructure as a Service* (IaaS): Amazon EC2.
- Tipos de nuvem:
  - Pública: disponível ao público em geral, cobrada conforme utilização.
  - Privada: não disponível ao público em geral (interna a uma organização).

Armbrust, M. et al. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50-58.

# Table of Contents

Introdução

Experimentos Científicos

Workflows Científicos

Parsl

Mão na massa

Conclusão

Laboratório Nacional de Computação Científica

# Workflows Científicos

- Um **workflow científico** consiste da especificação de um encadeamento de aplicações científicas a serem executadas e de dependências mútuas.
- Segue um ciclo de vida análogo ao dos experimentos científicos computacionais:
  - Composição, representação e modelagem de dados.
  - Mapeamento e execução.
  - Coleta de metadados e proveniência.
- Um **sistema de gerência de workflows científicos** (SGWC) permite gerenciar o ciclo de vida de workflows científicos.

Liu, J., Pacitti, E., Valduriez, P., Mattoso, M. (2015). A Survey of Data-Intensive Scientific Workflow Management. Journal of Grid Computing, 13(4), 457–493.

- **Sistemas de gerência de workflows científicos** (SGWC) visam a automação de experimentos científicos computacionais:
  - escalonamento de tarefas baseado em dependências de dados;
  - fluxo de dados entre tarefas;
  - execução paralela de tarefas independentes;
  - escalonamento de tarefas em ambientes de computação de alto desempenho;
  - gerência e consulta de dados de proveniência.

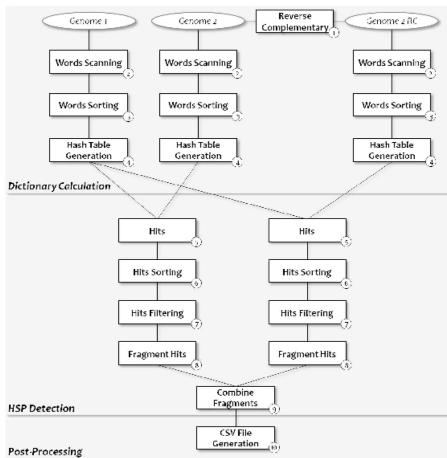
## Exemplo: Laboratório de Bioinformática, LNCC

Sequenciador Roche 454 FLX gera 12GB a 15GB por rodada de sequenciamento.
Dados processados por várias aplicações:

- filtragem por qualidade de dados gerados pelo sequenciador.
- formatação de dados.
- comparação das sequências com bases de dados conhecidas.

Laboratório Nacional de Computação Científica

# Exemplo: SwiftGECKO (genômica comparativa)




Mondelli, M. L., ..., Gadelha, L. M. R. (2018). BioWorkbench: A High-Performance Framework for Managing and Analyzing Bioinformatics Experiments. arXiv:1801.03915.
http://arxiv.org/abs/1801.03915

Laboratório Nacional de Computação Científica

# Workflow *desideratum*

- Clareza
- Facilidade na criação de fluxos corretos (corretude)
- Facilidade na verificação dos resultados
- Reportabilidade
- Reusabilidade
- Modelagem de dados
- Otimização automática

McPhillips, T., et al. "Scientific workflow design for mere mortals", Future Generation Comp Sy, v. 25, n. 5, pp. 541–551, 2009.

Laboratório Nacional de Computação Científica

# Pesquisa em Workflows Científicos

- Acoplamento de tarefas e transferência de dados entre tarefas de um workflow.
- Modelos de programação, interface com o usuário, comunicação entre tarefas e portabilidade.
- Monitoramento: andamento de execução, algoritmos para detecção de anomalias.
- Validação de execução de workflow:
  - reproduzir um workflow no mesmo ou em outro ambiente computacional,
  - comparar a execução com modelos de desempenho e com a proveniência coletada durante a execução,
  - comparar os resultados científicos com o que era esperado.

Deelman, E. et al. (2018). The future of scientific workflows. The International Journal of High Performance Computing Applications, 32(1), 159–175. https://doi.org/10.1177/1094342017704893

Laboratório Nacional de Computação Científica

# Workflows Científicos

- Workflows científicos típicos:
  - *Bag of tasks* (MG-RAST, DOCK),
  - Múltiplos estágios que usam arquivos como dados intermediários (Montage, BLAST),
  - Aplicações distribuídas com pares chave-valor intermediários (histogramas de dados para física de altas energias),
  - Cadeias de tarefas do tipo MapReduce (mineração de grafos),
  - Aplicações iterativas com variação no número de tarefas (otimização, filtros de Kalman),

Deelman, E. et al. (2018). The future of scientific workflows. The International Journal of High Performance Computing Applications, 32(1), 159–175. https://doi.org/10.1177/1094342017704893

# Workflows Científicos

- Workflows científicos podem ser executados *in situ* ou de forma distribuída.
- No caso *in situ*:
  - Processamento de dados, triagem, filtragem, análise ou visualização podem ocorrer enquanto o workflow está executando.
  - Essas ações ocorrem antes de se mover dados para fora de um supercomputador para análises adicionais
- Não estão disponíveis ferramentas de workflow de propósito geral amplamente adotadas que funcionem tanto *in situ* como em ambientes distribuídos.

Deelman, E. et al. (2018). The future of scientific workflows. The International Journal of High Performance Computing Applications, 32(1), 159–175. `https://doi.org/10.1177/1094342017704893`

Laboratório Nacional de Computação Científica

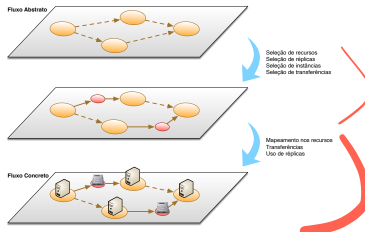
- Motivações para workflows *in situ*:
  - HPC
  - minimização da movimentação de dados através da exploração da localidade de dados, processando os dados *in place*
  - apoiar análises interativas (*human in the loop*),
  - captura de proveniência para suportar a análise interativa e permitir execução adaptativa do experimento (*user steering*)

Deelman, E. et al. (2018). The future of scientific workflows. The International Journal of High Performance Computing Applications, 32(1), 159–175. https://doi.org/10.1177/1094342017704893

# Composição de Workflows Científicos

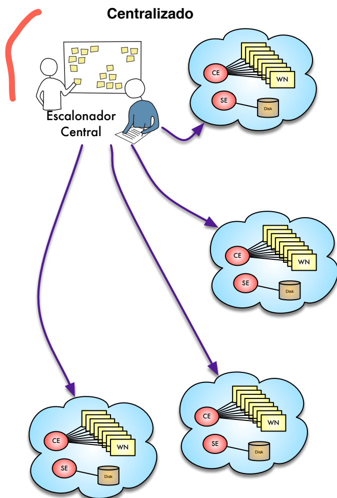
Etapa em que são definidas as aplicações componentes e as dependências de dados. Níveis de especificação:

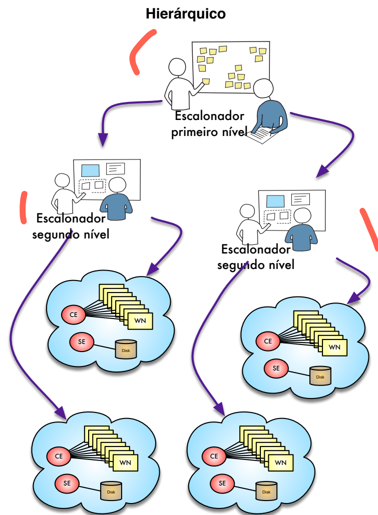
- Abstrato: tarefas abstratas, descritas por funcionalidade geral (p. ex., comparação de seqüências)
- Concreto: aplicações científicas e conjuntos de dados específicos



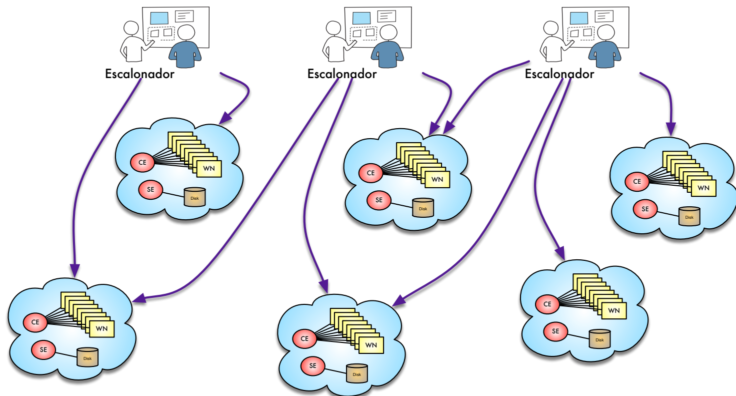

Laboratório Nacional de Computação Científica

# Sequenciamento (1/2)





# Sequenciamento (2/2)




Laboratório Nacional de Computação Científica

- Representação:
  - Textual: linguagem de programação.
  - Gráfica: interface gráfica onde nós são tarefas e arestas são dependências.

# Exemplo: Composição Textual

```
type fastaseq;
type headerfile;
type indexfile;
type seqfile;
type database
{
  headerfile phr;
  indexfile pin;
  seqfile psq;
}
type query;
type output;
string num_partitions=@arg("n", "8");
string program_name=@arg("p", "blastp");
fastaseq dbin <single_file_mapper;file=@arg("d", "database")>;
query query_file <single_file_mapper;file=@arg("i", "sequence.seq")>;
string expectation_value=@arg("e", "0.1");
output blast_output_file <single_file_mapper;file=@arg("o",
                                                     "output.html")>;
string filter_query_sequence=@arg("F", "F");
fastaseq partition[] <ext;exec="splitmapper.sh",n=num_partitions>;

app (fastaseq out[]) split_database (fastaseq d, string n)
{
  fastasplitn @filename(d) n;
}

app (database out) formatdb (fastaseq i)
{
  formatdb "-i" @filename(i);
}

app (output o) blastapp(query i, fastaseq d, string p, string e, string f,
                        database db)
{
  blastall "-p" p "-i" @filename(i) "-d" @filename(d) "-o" @filename(o)
           "-e" e "-T" "-F" f;
}

app (output o) blastmerge(output o_frags[])
{
  blastmerge @filename(o) @filenames(o_frags);
}

partition=split_database(dbin, num_partitions);

database formatdbout[] <ext; exec="formatdbmapper.sh",n=num_partitions>;
output out[] <ext; exec="outputmapper.sh",n=num_partitions>;

foreach part,i in partition {
  formatdbout[i] = formatdb(part);
  out[i]=blastapp(query_file, part, program_name, expectation_value,
                  filter_query_sequence, formatdbout[i]);
}

blast_output_file=blastmerge(out);
```

Laboratório Nacional de Computação Científica

# Exemplo: Composição Gráfica

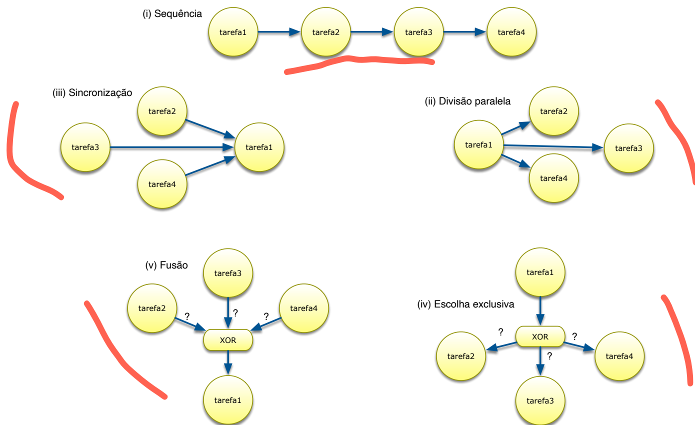
Taverna Workbench 2.1.0

Fonte: Taverna (`http://www.taverna.org.uk`)

Laboratório Nacional de Computação Científica

# Composição de Workflows científicos: Padrões (1/2)

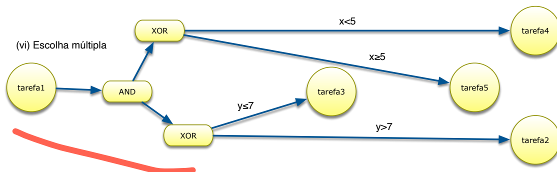
Foram identificados 43 padrões de composição de workflows.

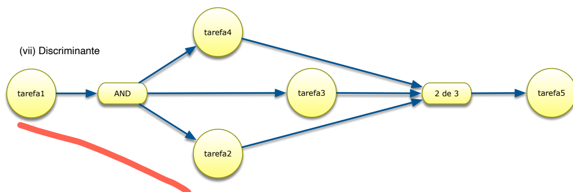



W. van der Aaalst et al. Workflow Patterns. *Distributed and Parallel Databases* 14(1):5-51, 2003.

Laboratório Nacional de Computação Científica

# Composição de Workflows científicos: Padrões (2/2)

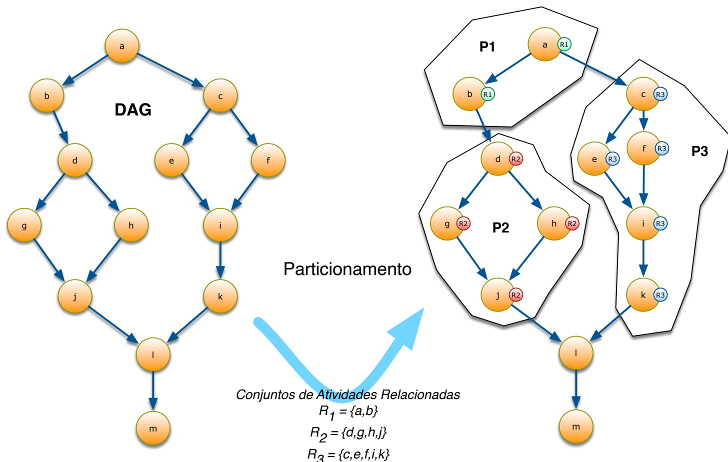



Fonte: D.Carvalho. Controle distribuído de Workflows em malhas computacionais. Rio de Janeiro: UFRJ/COPPE, 2010.

Laboratório Nacional de Computação Científica

# Exemplo: Abstrato .vs. Concreto

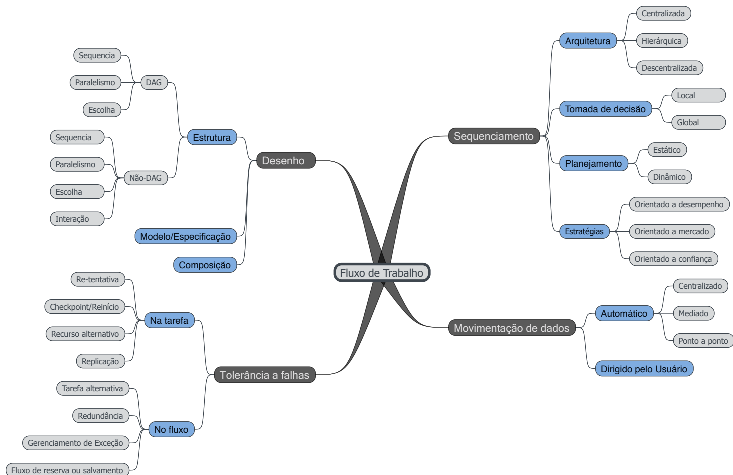



Fonte: D.Carvalho. Controle distribuído de Workflows em malhas computacionais. Rio de Janeiro: UFRJ/COPPE, 2010.

# Coleta de metadados e proveniência

- Coleta de eventos do ciclo de vida do workflow:
  - Mudanças e evolução da especificação.
  - Consumo e produção de dados por aplicações componentes executadas.
- Gerência de metadados relacionados ao domínio científico (semântica do experimento).
- Serviço de consulta a metadados e proveniência.
- Objetivos: reprodutibilidade, verificação, análise.

# Taxonomia dos Workflows




Fonte: D.Carvalho. Controle distribuído de Workflows em malhas computacionais. Rio de Janeiro: UFRJ/COPPE, 2010.

Laboratório Nacional de Computação Científica

# Table of Contents

Introdução

Experimentos Científicos

Workflows Científicos

Parsl

Mão na massa

Conclusão

Laboratório Nacional de Computação Científica

# Parsl - Visão geral (1/4)

- Parsl é uma biblioteca para o Python que permite gerenciar workflows científicos paralelos.
- O paralelismo é implícito e baseado nas dependências das aplicações componentes do workflow.
- O motor de execução inclui diversas opções de execução, desde multithreading local a execução em supercomputadores com Swift/T.


http://parsl-project.org/.

Babuji, Y. et al. (2018). Parsl : Scalable Parallel Scripting in Python. In 10th International Workshop on Science Gateways (IWSG 2018).

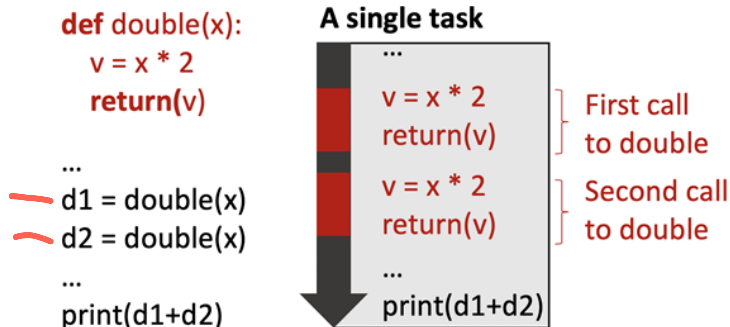

Laboratório Nacional de Computação Científica

# Parsl - Visão geral (2/4)

Parsl é desenhado para fazer a orquestração de tarefas assíncronas de workflows em Python. Ele gerencia a execução das tarefas através de diversos recursos computacionais (laptops até supercomputadores).

- as tarefas são funções escritas em Python "com anotações" que indicam que elas são execução concorrente
- usa código Python padrão para invocar as funções anotadas e garante as dependências do fluxo através Futures e Data-Futures
- Futures e Data-Futures são conceitos fundamentais

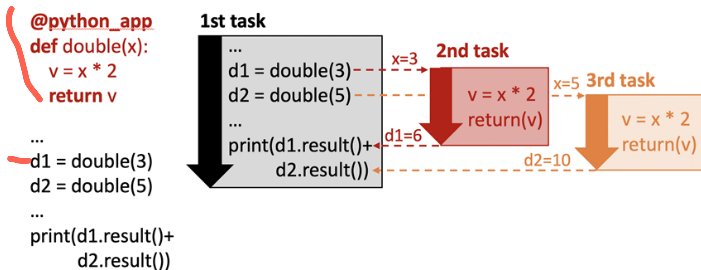
# Parsl - Visão geral (3/4)




Fonte: `https://parsl.readthedocs.io/en/stable/userguide/`

Laboratório Nacional de Computação Científica

# Parsl - Visão geral (4/4)




Fonte: `https://parsl.readthedocs.io/en/stable/userguide/`

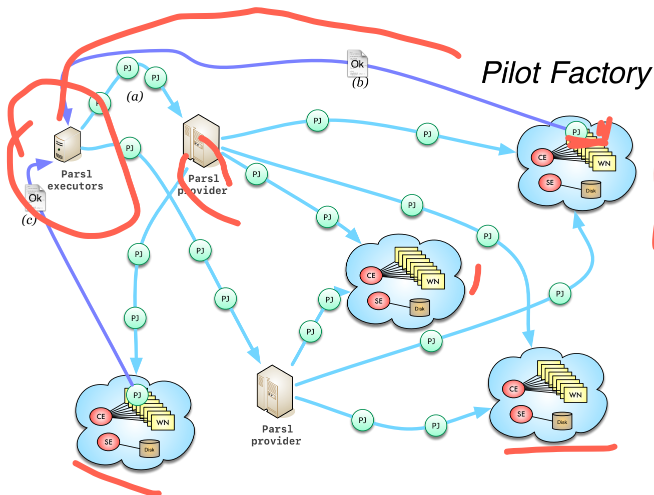
Laboratório Nacional de Computação Científica

# A execução do Workflow

- As tarefas Parsl são executadas em conjunto com o programa principal (em Python)
- Os `parsl.executors` fornecem os recursos para o Parsl sequenciar o workflow
- Os recursos podem ser locais ou remotos
- Alguns `parsl.executors` (`HighThroughputExecutor`) implementam a execução através de *Pilot Jobs*

Laboratório Nacional de Computação Científica

# Pilot Jobs




Fonte: D.Carvalho. Controle distribuído de Workflows em malhas computacionais. Rio de Janeiro: UFRJ/COPPE, 2010.

Laboratório Nacional de Computação Científica

# Instalação do Parsl

- Tutorial baseado em `https://parsl.readthedocs.io`
- Parsl requer Python $\geq 3.5$.

```
$ pip3 install parsl
$ pip3 install jupyter
```

- No Santos Dumont

```
$ module add python/3.8.2
```

Laboratório Nacional de Computação Científica

# Verificando a versão do Parsl

```
$ python3
>>> import parsl
>>> from parsl.app.app import python_app, bash_app
>>> from parsl.configs.local_threads import config
>>> print(parsl.__version__)

1.0.0
```

Laboratório Nacional de Computação Científica

# Carregando a configuração

- O Parsl permite expressar como o workflow é composto.
- Os detalhes de tempo de execução devem ser carregados de um arquivo de configuração.

```
>>> from parsl.configs.local_threads import config
>>> parsl.load(config)

<parsl.dataflow.dflow.DataFlowKernel at 0x7fed94bf32b0
    >
```

Laboratório Nacional de Computação Científica

# Futures e Data Futures

- Baseados em *future, promise, delay, and deferred theory(*)*
- Os valores de retorno e parâmetros também tem por objetivo a sincronização
- Mesma ideia do Python `concurrent.futures` (introduzido no Python 3.2)
- Tem valor especial (não atribuído) até o que representa seja concretizado
- O método result() bloqueia a tarefa (ou aplicação principal) se o resultado ainda não estiver pronto
- O método done() verifica se o resultado já está disponível (retorna `True` ou `False`)

Promise in 1976 by Daniel P. Friedman and David Wise.

Eventual in 1976 by Peter Hibbard.

Future in 1977 by Henry Baker and Carl Hewitt.

Laboratório Nacional de Computação Científica

# Com Futures

```
@python_app
def double(x):
    v = x * 2
    return v

...
d1 = double(3)
d2 = double(5)
...
print(d1.result()+
      d2.result())
```

**1st task**

```
...
d1 = double(3)
d2 = double(5)
...
print(d1.result()+
      d2.result())
```

x=3

**2nd task**

```
v = x * 2
return(v)
```

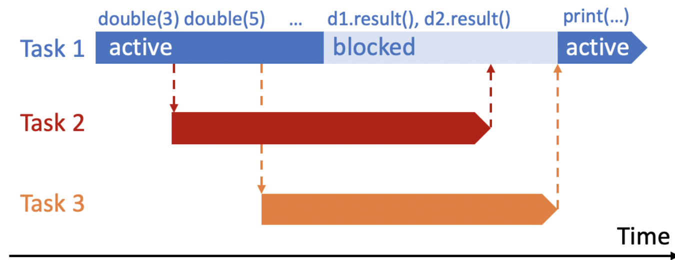
d1=6

x=5

**3rd task**

```
v = x * 2
return(v)
```

d2=10




Fonte: `https://parsl.readthedocs.io/en/stable/userguide/`

# Apps

- A especificação das tarefas em Parsl é feita através de Apps.
- Parsl provê suporte aos seguintes tipos de Apps:
  - código nativo em Python (`python_app`),
  - programas executados via de linha de comando (`bash_app`).

Laboratório Nacional de Computação Científica

# Parsl Python Apps

- Uma App do tipo Python é declarado usando o decorador `@python_app`.

```
@python_app
def hello ():
    return 'Hello World!'

print(hello().result())
```

```
@python_app
def multiply (a, b):
    return a * b

print(multiply(5,9).result())
```

Laboratório Nacional de Computação Científica

- Apps Python podem ser executadas em recursos remotos, as dependências devem ser carregadas na definição da App.

```
@python_app
def slow_hello():
    import time
    time.sleep(5)
    return 'Hello World!'

print(slow_hello().result())
```

# Parsl Python Apps

- tarefas do Parsl, escritas em Python e que se comunicam com o programa Parsl principal através de parâmetros e valores de retorno (função)
- se os parâmetros forem Futures, dependências automáticas são criadas
- podem se comunicar através de arquivos (parâmetros especiais)

```
# Omissis...
@python_app
def echo(inputs=[], outputs=[]):
    with open(inputs[0], 'r') as in_file, open(outputs
    [0], 'w') as out_file:
        out_file.write(in_file.readline())

echo(inputs=['in.txt'], outputs=['out.txt'])
```

Laboratório Nacional de Computação Científica

# Parsl Bash App

- Uma App do tipo Python é declarado usando o decorador @bash_app.

```
@bash_app
def echo_hello(stdout='echo-hello.stdout',
               stderr='echo-hello.stderr'):
    return 'echo "Hello World!"'

echo_hello().result()

with open('echo-hello.stdout', 'r') as f:
     print(f.read())
```

# Parsl Bash App

- tarefas do Parsl para aplicações externas
- não retornam valor, sincronização via result() e done()
- podem se comunicar através de arquivos (parâmetros especiais)
- dependências automáticas são criadas em Futures

```
# Omissis...
@bash_app
def echo(arg, inputs=[], stderr=parsl.AUTO_LOGNAME,
    stdout=parsl.AUTO_LOGNAME):
    return f'echo {arg} {inputs[0]} {inputs[1]}'

future = echo('Hello', inputs=['World', '!'])
future.result() # block until task has completed

with open(future.stdout, 'r') as f:
    print(f.read()) # prints "Hello World !"
```

# Passagem de parâmetros para Apps

Parâmetros podem ser tipos primitivos e objetos serializáveis (i.e., Numpy Arrays, Pandas DataFrames, marshalable objetos).

```
import parsl
from parsl import python_app

parsl.load()

@python_app
def double(x):
    return x * 2

# invoke the Python app and print the result (8)
print(double(2).result()+double(2).result())
```

Laboratório Nacional de Computação Científica

# Passando dados para as Apps

- As palavras-chaves `inputs` e `outputs` podem ser usadas nas declarações das Apps para expressar as dependências de dados.
- Vamos supor que geramos três arquivos de texto:

```
$ echo "hello 1" > /tmp/hello1.txt
$ echo "hello 2" > /tmp/hello2.txt
$ echo "hello 3" > /tmp/hello3.txt
```

Laboratório Nacional de Computação Científica

# Passando dados para as Apps

- Eles podem ser processados pelo App cat da seguinte forma:

```
@bash_app
def cat(inputs=[], outputs=[]):
    return 'cat %s > %s' %(" ".join(inputs), outputs
    [0])

concat = cat(inputs=['/tmp/hello1.txt','/tmp/hello2.
   txt','/tmp/hello3.txt'],
             outputs=[File('all_hellos.txt')])

# Open the concatenated file
with open(concat.outputs[0].result().filepath, 'r') as
     f:
     print(f.read())
```

Laboratório Nacional de Computação Científica

# DataFutures

- Cada chamada a uma App pode especificar outputs, que são DataFutures.
- DataFutures são monitorados para garantir que sejam criados e passados para Apps que dependem deles.

```
@bash_app
def slowecho(message, outputs=[]):
    return 'sleep 5; echo %s &> {outputs[0]}' % (
    message)

hello = slowecho('Hello World!',
        outputs=[os.path.join(os.getcwd(), 'hello-
    world.txt')])

print(hello.outputs)
[<DataFuture at 0x10e0b29b0 state=running>]

print(hello.outputs)
[<DataFuture at ... state=finished returned hello-
    world.txt>]
```

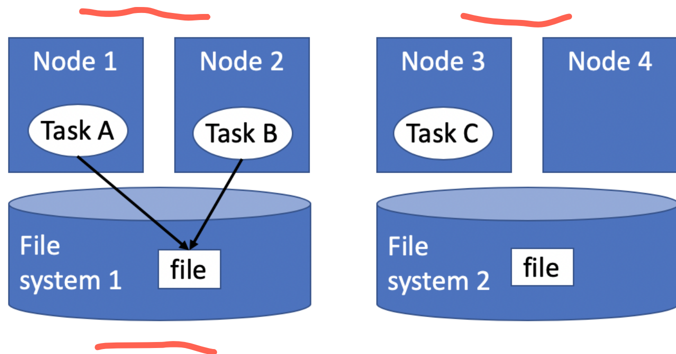
Laboratório Nacional de Computação Científica

Memória e *Name space* não são necessariamente compartilhados entre a aplicação principal python e as tarefas (`ThreadPoolExecutor` vs `HighThroughputExecutor`)

```
# Omissis...

factor = 5

@python_app
def good_double(factor, x):
    import random
    return x * random.random() * factor

print(good_double(factor, 42).result())
```

Laboratório Nacional de Computação Científica

# Arquivos Parsl




Fonte: https://parsl.readthedocs.io/en/stable/userguide/

Laboratório Nacional de Computação Científica

# Arquivos Parsl

- Parsl pode utilizar arquivos em tarefas Parsl como unidade de comunicação entre tarefas através de parâmetros de entrada e saída
- Esses arquivos são interessantes quando:
  - os dados trocados não são serializáveis em objetos Python
  - os dados são de tamanho significativo
  - as tarefas são desenhadas para utilizar arquivos como entrada e saída
- os `parsl.data_provider.files.File` são independentes de localização
- quando usados como parâmetros de entrada e saída, eles entram no grafo de dependências do workflow e são transmitidos antes da execução da tarefa

Laboratório Nacional de Computação Científica

# Parsl File

- permite referência e acesso independente de *fylesystem*
- pode ser FTP, HTTP, HTTPS: in-task staging, Globus, rsync

```
# Omissis...
@python_app
def print_file(inputs=[]):
    with open(inputs[0].filepath, 'r') as inp:
        content = inp.read()
        return(content)

# create an remote Parsl file
f = File('https://github.com/Parsl/parsl/blob/master/
    README.rst')

# call the print_file app with the Parsl file
r = print_file(inputs=[f])
r.result()
```

# Parsl Executors

Extende a classe `concurrent.futures` do Python

- ThreadPoolExecutor (multi-thread local)
- HighThroughputExecutor (hierárquico através de Pilot Jobs)
- WorkQueueExecutor (em Beta)
- ExtremeScaleExecutor (em Beta)

Laboratório Nacional de Computação Científica

# Parsl Execution Providers

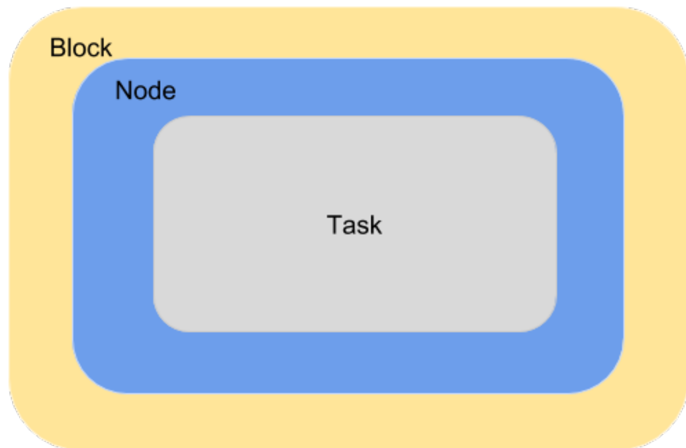
- LocalProvider
- CobaltProvider
- SlurmProvider
- CondorProvider
- GridEngineProvider
- TorqueProvider
- AWSProvider
- GoogleCloudProvider
- JetstreamProvider
- KubernetesProvider
- AdHocProvider
- LSFProvider

# Parsl Launchers

Implementa mecanismos para disparo de tarefas através dos nós pelos Executors

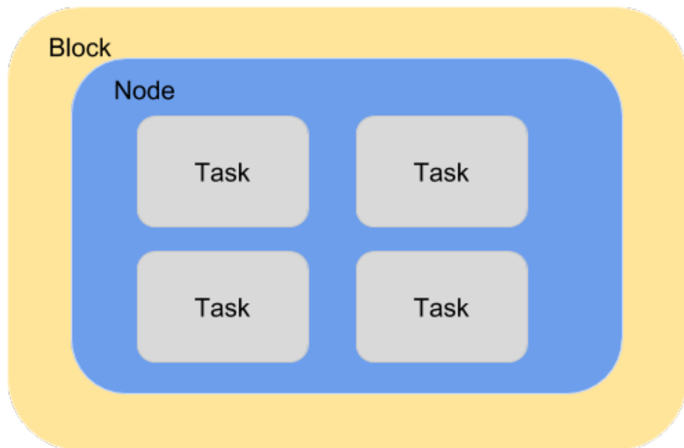
- SrunLauncher: (Srun based launcher para sistemas baseados em Slurm)
- AprunLauncher: (Aprun based launcher para Crays)
- SrunMPILauncher: (Para disparo de aplicações MPI em sistemas com Srun)
- GnuParallelLauncher: (usa GNU parallel)
- MpiExecLauncher: (implementa Mpiexec)
- SimpleLauncher: (default)
- SingleNodeLauncher: (implementa workers_per_node)

# Blocos Parsl




Fonte: `https://parsl.readthedocs.io/en/stable/userguide/`

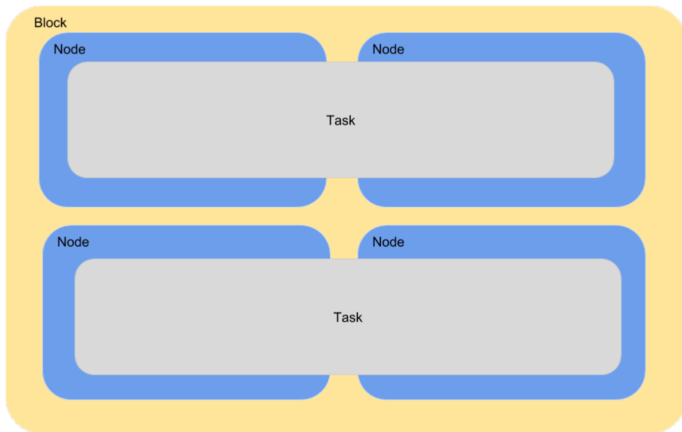
# Blocos Parsl




Fonte: `https://parsl.readthedocs.io/en/stable/userguide/`

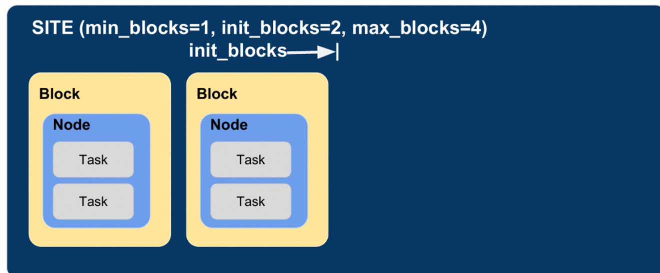
Laboratório Nacional de Computação Científica

# Blocos Parsl



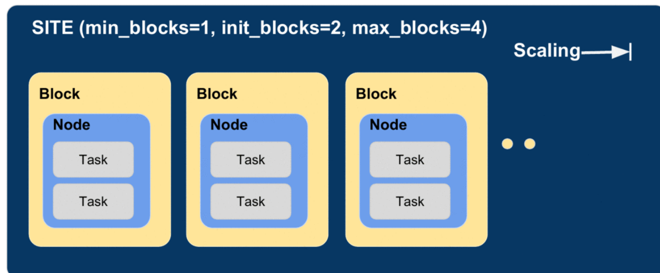

Fonte: `https://parsl.readthedocs.io/en/stable/userguide/`

# Elasticidade Parsl




Fonte: https://parsl.readthedocs.io/en/stable/userguide/

# Elasticidade Parsl




Fonte: `https://parsl.readthedocs.io/en/stable/userguide/`

# Configuração

- Configuração com múltiplas *threads* locais:

```
from parsl.config import Config
from parsl.executors.threads import ThreadPoolExecutor

local_threads = Config(
    executors=[
        ThreadPoolExecutor(
            max_threads=8,
            label='local_threads'
        )
    ]
)
```

Laboratório Nacional de Computação Científica

# Configuração

- Configuração com *job* piloto:

```
from parsl.providers import LocalProvider
from parsl.channels import LocalChannel
from parsl.config import Config
from parsl.executors import HighThroughputExecutor
local_htex = Config(
    executors=[
        HighThroughputExecutor(
            label="htex_Local",
            worker_debug=True,
            cores_per_worker=1,
            provider=LocalProvider(
                channel=LocalChannel(),
                init_blocks=1,
                max_blocks=1, ), ) ],
    strategy=None, )
```

Laboratório Nacional de Computação Científica

# Configuração

- Configuração para acesso remoto ao SLURM:

```
from parsl.providers import SlurmProvider
from parsl.channels import SSHChannel
from parsl.launchers import SrunLauncher
...
config = Config(
    executors=[
        IPyParallelExecutor(
            ...
            provider=SlurmProvider(
                'debug',
                channel=SSHChannel(
                    hostname='login.sdumont.lncc.br',
                    username='diego.carvalho',
                    ...
```

Laboratório Nacional de Computação Científica

# Error handling

Implementa *Lazy fail* e o workflow continua mesmo que uma tarefa falhou. Motivos de falha de um workflow Parsl:

- a tarefa falha (*Python exception* ou *return value* diferente de zero)
- Tarefa falhou no disparo por motivo de dependência (DAG)
- Erro no string de definição de uma Bash App
- A tarefa completou, mas não produziu um dos outputs
- A tarefa estourou o Walltime

Laboratório Nacional de Computação Científica

# Além do escopo de hoje

- App caching (results)
- App equivalence (code objects)
- Checkpointing
- Monitoring

Laboratório Nacional de Computação Científica

# Cálculo de $\pi$ com Monte Carlo

- Estimar $\pi$ jogando dardos em um quadrado unitário.
- Calcular o percentual que acerta um círculo unitário:
  - Área do quadrado: $r^2 = 1$.
  - Área do círculo no quadrante: $\frac{\pi r^2}{4} = \frac{\pi}{4}$.
- Jogando dardos randomicamente em posições $x$, $y$.
- Se $x^2 + y^2 < 1$, o ponto estará dentro do círculo.
- A proporção que acertará dentro ($\frac{\text{acertos}}{\text{tentativas}}$) esperada é $\frac{\text{acertos}}{\text{tentativas}} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$.
- Logo $\pi = 4\frac{\text{acertos}}{\text{tentativas}}$

Laboratório Nacional de Computação Científica

# Exemplo de workflow Parsl

```
@python_app
def pi(total):
    import random
    width = 10000
    center = width/2
    c2  = center**2
    count = 0
    for i in range(total):
        # Drop a random point in the box.
        x,y = random.randint(1, width),
              random.randint(1, width)
        # Count points within the circle
        if (x-center)**2 + (y-center)**2 < c2:
            count += 1
    return (count*4/total)
```

Laboratório Nacional de Computação Científica

# Exemplo de workflow Parsl

```
@python_app
def mysum(a,b,c):
    return (a+b+c)/3

a, b, c = pi(10**6), pi(10**6), pi(10**6)
avg_pi  = mysum(a, b, c)
```

Laboratório Nacional de Computação Científica

# Table of Contents

Introdução

Experimentos Científicos

Workflows Científicos

Parsl

Mão na massa

Conclusão

Voltamos 17h

Laboratório Nacional de Computação Científica

# O que vamos fazer

- Alguns exemplos do tipo: Hello Workflow
- Padrões
  - Bag of tasks
  - Sequencial workflows
  - Parallel workflows
  - Parallel workflows com loops
  - MapReduce
- Exemplos de configuração
- Exemplos no Santos Dumont (sequenciamento hierárquico)

# Table of Contents

Introdução

Experimentos Científicos

Workflows Científicos

Parsl

Mão na massa

Conclusão

Laboratório Nacional de Computação Científica

# Conclusão

- Uma introdução de e-Science e Workflows
- Workflows
- Parsl
- Mão na massa

Laboratório Nacional de Computação Científica

Obrigado!

E-mail: `d.carvalho@ieee.org`

```
git clone git@github.com:diegomcarvalho/2021-ProgramaVeraoSD.git
```

Laboratório Nacional de Computação Científica