Flowbster: Dynamic creation of data pipelines in clouds

Peter Kacsuk, Jozsef Kovacs and Zoltan Farkas

MTA SZTAKI

kacsuk@sztaki.hu

Motivations for Flowbster

- Processing big data typically means to execute a set of tasks on a large data set
- These tasks are typically executed in a certain order
- Such an ordering of tasks can be represented as a Dataflow graph
- Such dataflow graphs can be executed by dataflow workflow systems
- The goal is to execute such dataflow workflow systems in clouds using as many cloud resources as needed (on-demand resource usage)
- The name of this new workflow system is **Flowbster**

Flowbster contra job-oriented workflow systems

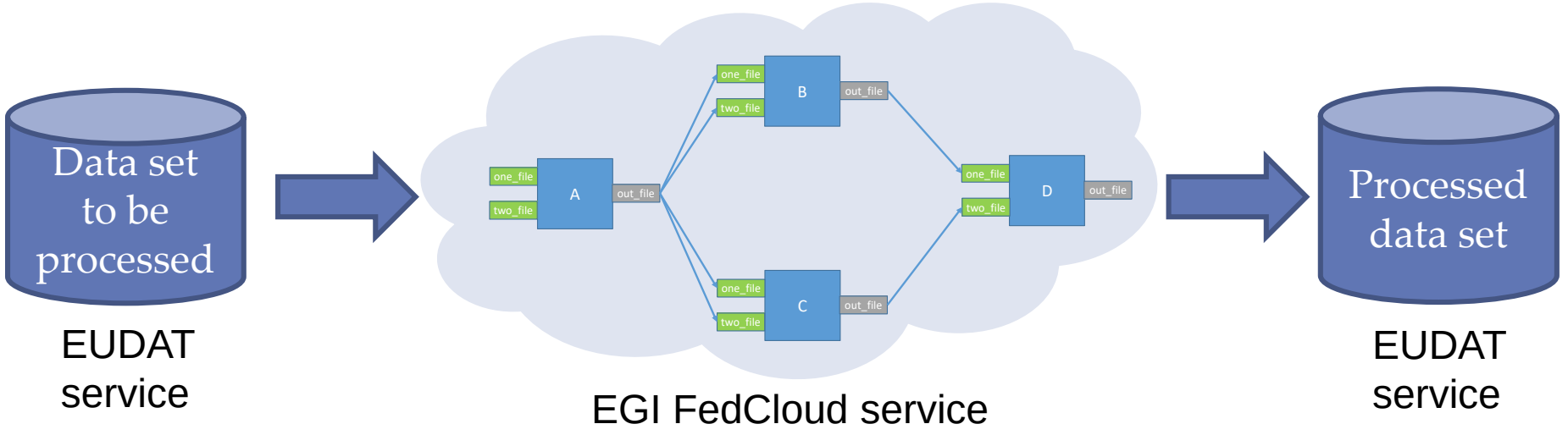
- Job-oriented workflow systems work based on service orchestration
- Nodes of a workflow represent jobs to be executed in the infrastructure
- There is a workflow enactor (orchestrator) that recognizes that a certain node/job can be executed and submits this job together with the required data
- The result data is typically transferred back to the enactor (or its storage)
- This execution mechanism is not optimal, requires to much data transfer

Flowbster contra job-oriented workflow systems

- In Flowbster there is no enactor, it works based on **service coreography**
- Nodes of the workflow directly communicate the data among them
- Data is passed through the workflow as a **data stream**
- A node is activated and executes the assigned task when all the input data arrived
- There is no useless data transfer
- Nodes of Flowbster workflows are deployed in the cloud as VMs and they exist until all the input data sets are processed
- As a result a Flowbster workflow works as a temporary **virtual infrastructure deployed in the cloud**
- Input data sets flow through this virtual infrastructure and meanwhile they flow through they are processed by the nodes of the workflow

Concept of Flowbster

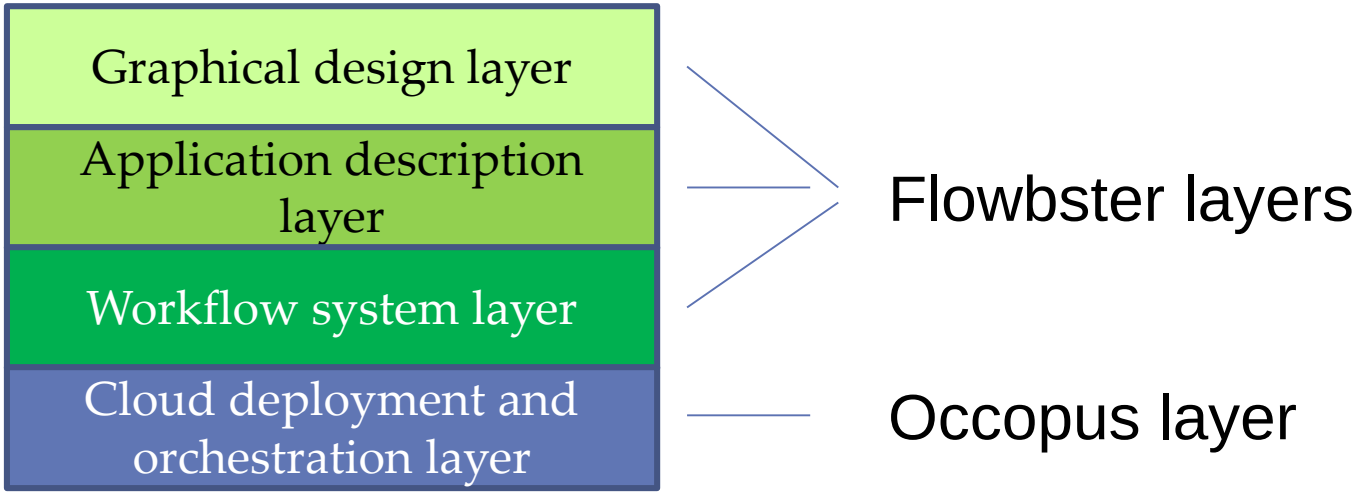
- The goal of Flowbster is to enable
  - The quick deployment of the workflow as a pipeline infrastructure in the cloud
  - Once the pipeline infrastructure is created in the cloud it is activated and data elements of the data set to be processed flow through the pipeline
  - As the data set flows through the pipeline its data elements are processed as defined by the Flowbster workflow



Structure of the Flowbster workflow system

MTA SZTAKI
Hungarian Academy of Sciences
Institute for Computer Science and Control

- Goal:
  - To create the Flowbster workflow in the cloud without any cloud knowledge
- Solution:
  - To provide a layered concept where users with different expertise can enter to the use of Flowbster
- 4 layers:



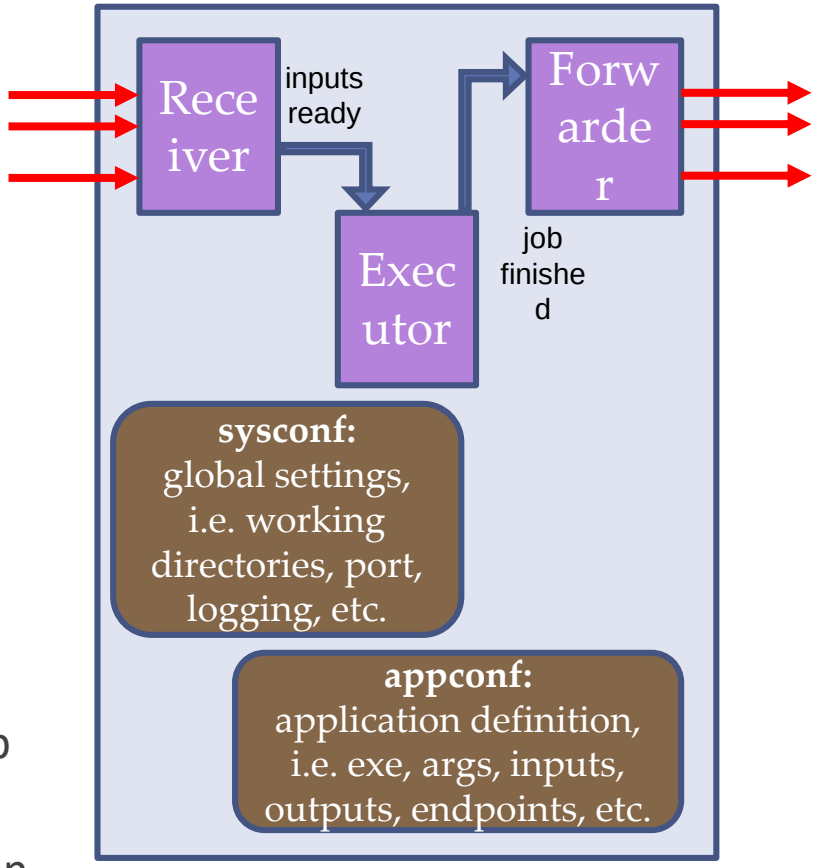
Occopus Layer

- Occopus is a cloud orchestrator and manager tool
- It automatically deploys virtual infrastructures (like Flowbster workflows) in the cloud based on an Occopus descriptor that consists of:
  - **Virtual infrastructure description**:
    - Specifies the **nodes** (services) to be deployed and all **cloud-independent** attributes e.g. input values for a service.
    - Specifies the **dependencies** among the nodes, to decide the order of deployment
    - Specifies **scaling** related attributes like min, max number of instances
  - **Node definition:**
    - Defines **how to construct the node** on a target cloud. This contains all **cloud dependent** settings, e.g. image id, flavour, contextualization
- See detailed tutorials at the Occopus web page:
  - http://occopus.lpds.sztaki.hu/tutorials

Flowbster Workflow System Layer

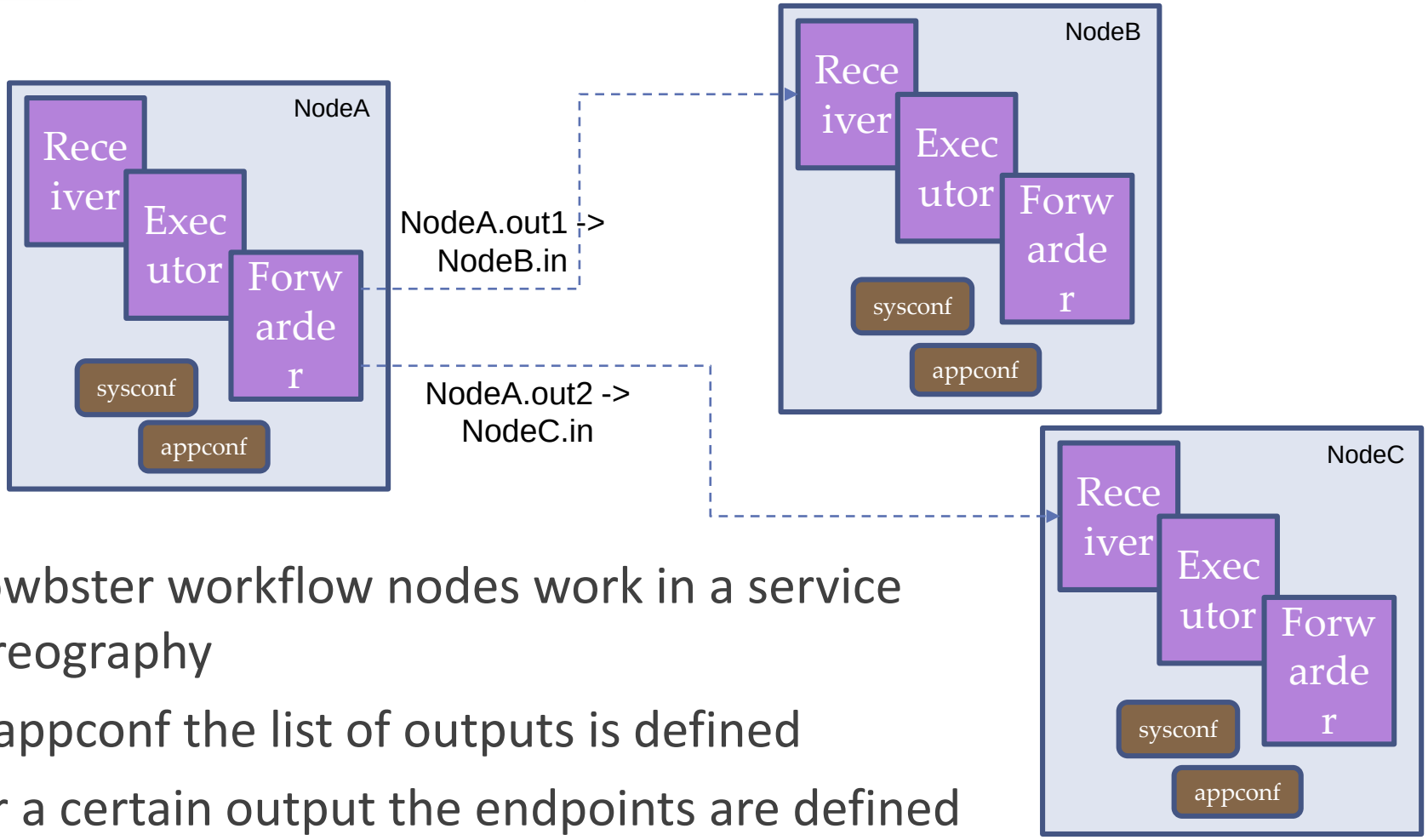
MTA SZTAKI
Hungarian Academy of Sciences
Institute for Computer Science and Control

- Contains uniform Flowbster workflow nodes which have the internal structure shown in the figure
- Every node provides the following actions:
  - Receives and keeps track of the input items
  - Executes the (pre-) configured application when inputs are ready
  - Identifies and forwards results of execution towards a (pre-) configured endpoint
- Contains 3 components:
  - Receiver: service to receive inputs
  - Executor: service to execute predefined app
  - Forwarder: service to send results of the finished app to a predefined remote location



Also requires 2 config files in order to costumize the node according to the workflow definition

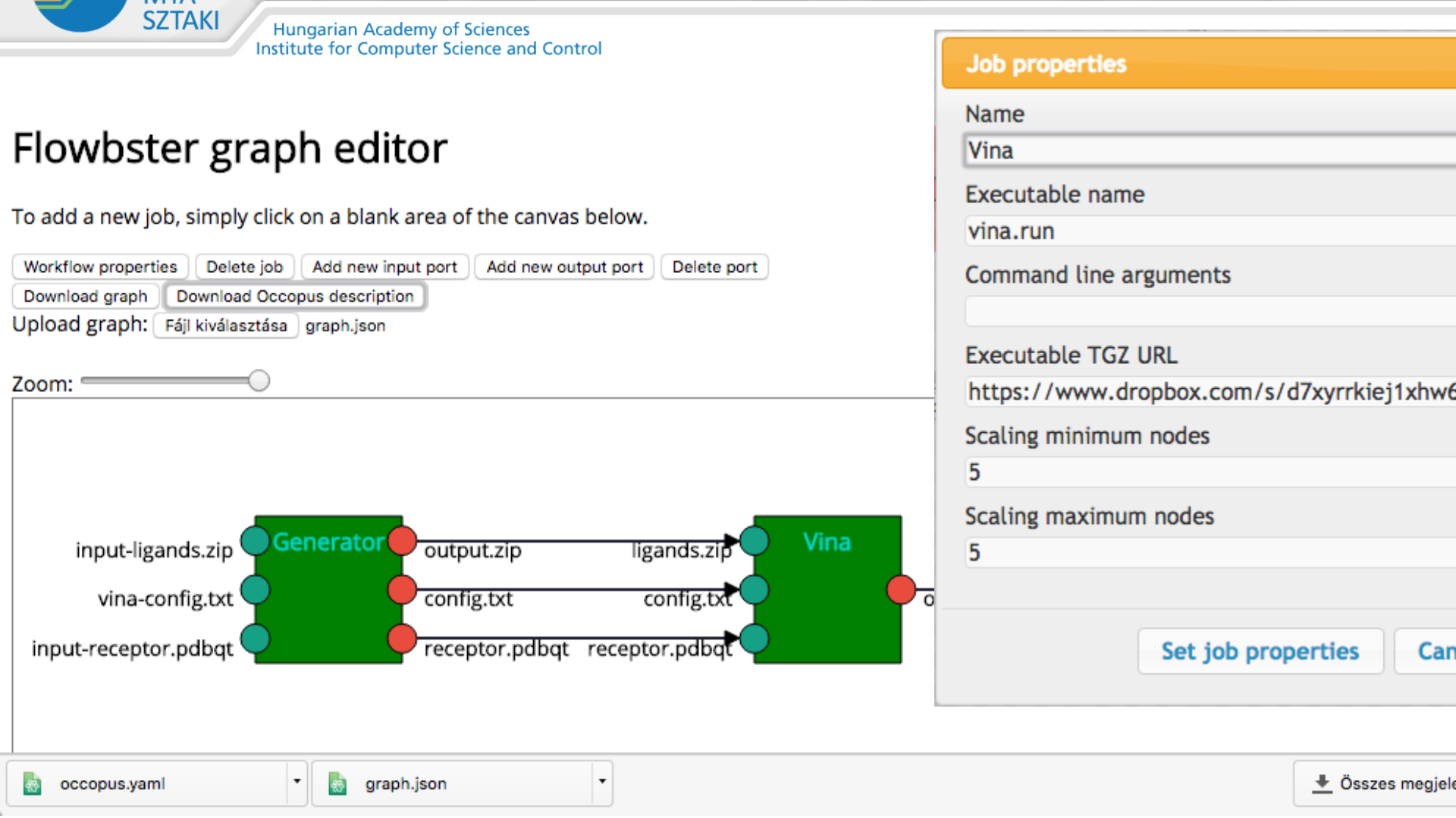
Connecting Flowbster nodes into a workflow



- Flowbster workflow nodes work in a service coreography
- In appconf the list of outputs is defined
- For a certain output the endpoints are defined
- An endpoint must point to a receiver node

Flowbster Graphical Design Layer

MTA SZTAKI
Hungarian Academy of Sciences
Institute for Computer Science and Control



Flowbster Graphical D

MTA SZTAKI

Hungarian Academy of Sciences
Institute for Computer Science and Control

Flowbster graph editor

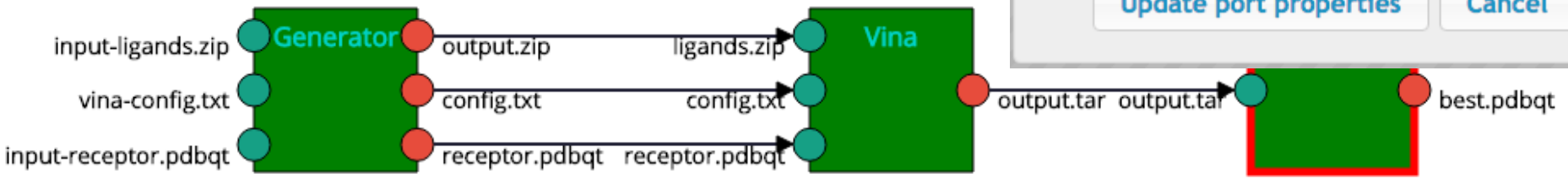
To add a new job, simply click on a blank area of the canvas below.

Workflow properties | Delete job | Add new input port | Add new output port | Delete port

Download graph | Download Occopus description

Upload graph: Fájl kiválasztása graph.json

Zoom:



occopus.yaml | graph.json | Összes megjele

Port properties

Name: output.tar

Target IP

Target port

Generator port

Filter regexp

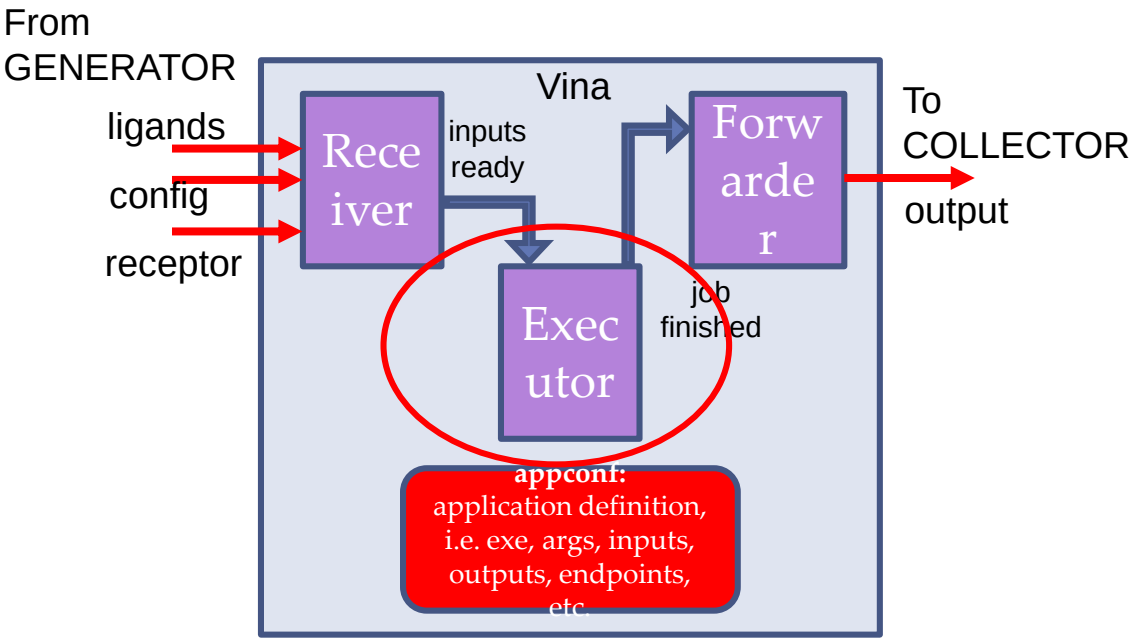
Distribution

Update port properties | Cancel

Flowbster Application Description Layer

MTA SZTAKI
Hungarian Academy of Sciences
Institute for Computer Science and Control

- **Automatically generated from the graphical view**
- It contains the Occopus descriptor of the Flowbster workflow
  - Virtual infrastructure descriptor representing the workflow graph
  - Customized node definitions for each node of the workflow. E.g. Vina node:

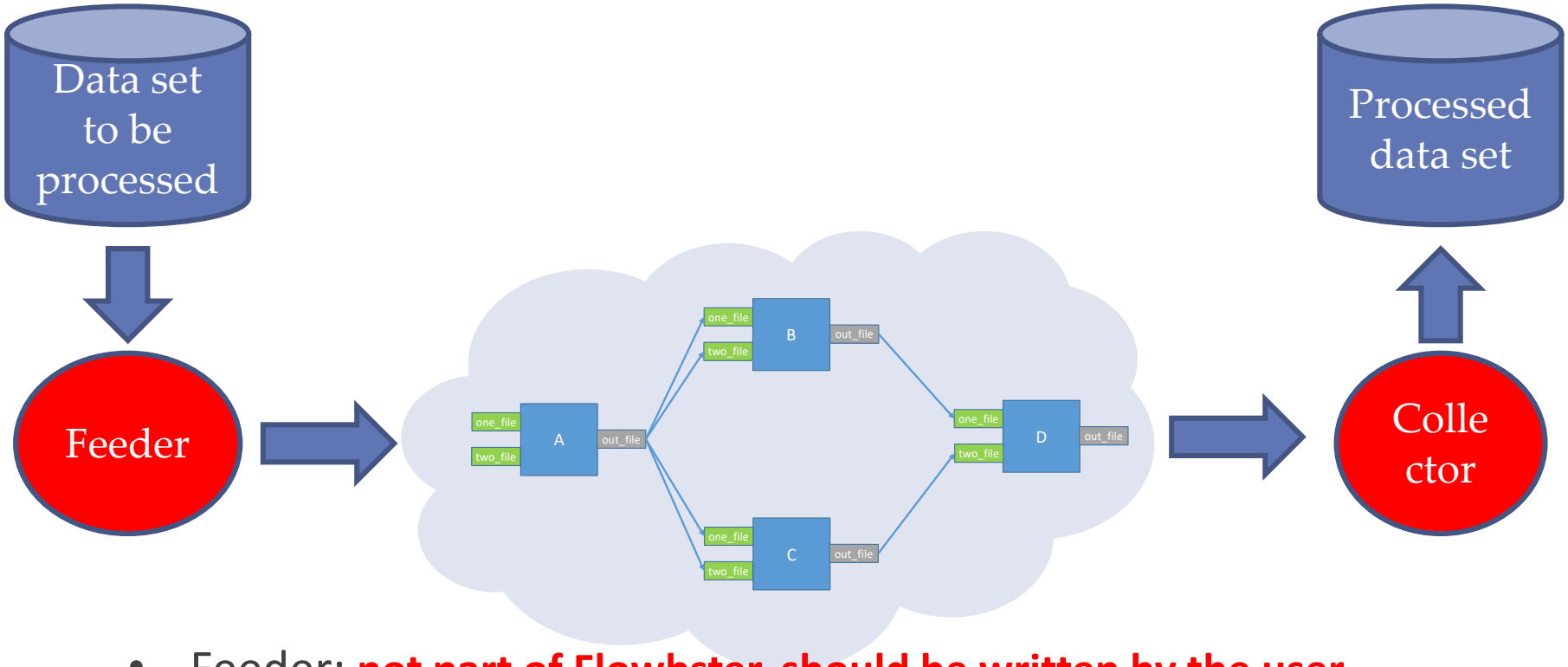



```
- &Vina
    name: Vina
    type: flowbster_node
    scaling:
        min: 5
        max: 5
    variables:
      jobflow:
        app:
          exe:
            filename: vina.run
            tgzurl: http://foo.bar/vina.tgz
          args: "
          in:
            -
              name: ligands.zip
            -
              name: config.txt
            -
              name: receptor.pdbqt
          out:
            -
              name: output.tar
              targetname: output.tar
              targetnode: COLLECTOR
```

Define parallellism

Feeding and gathering data set elements

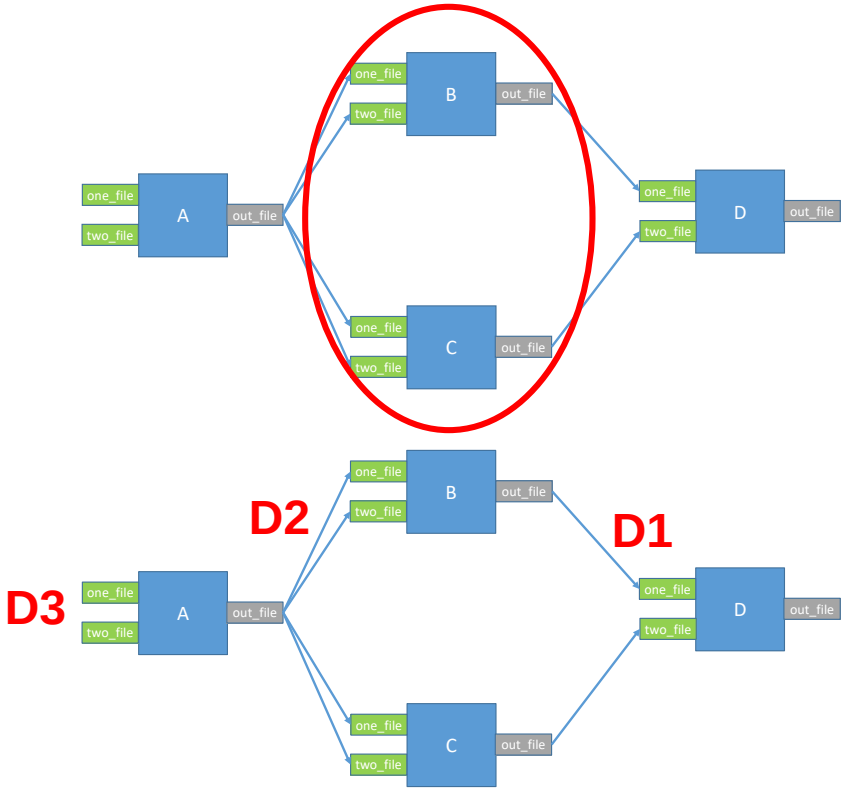
MTA SZTAKI
Hungarian Academy of Sciences
Institute for Computer Science and Control



- Feeder: **not part of Flowbster, should be written by the user**
  - Command line tool
  - Feeds a given node/port of Flowbster workflow with input data items
- Collector: **not part of Flowbster, should be written by the user**
  - Web service acting as a receiver
  - Transfers the incoming data items into the target storage

Exploitable parallelisms in Flowbster

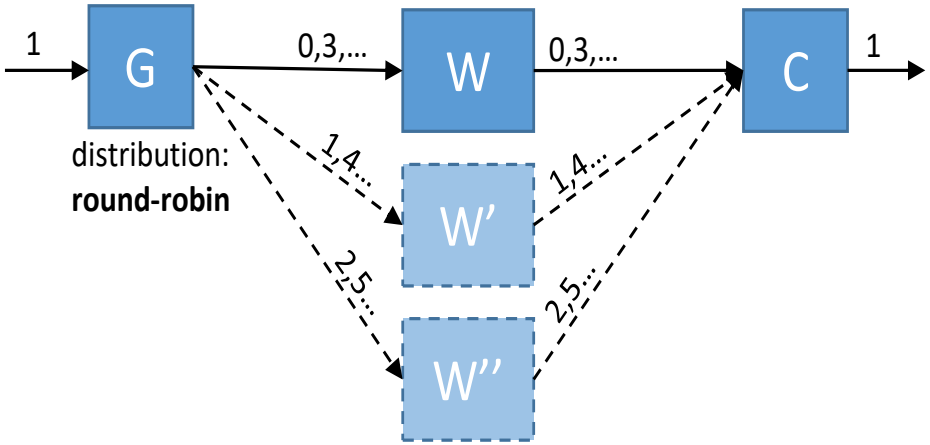
- **Parallel branch parallelism**
- **Pipeline parallelism**
- **Node scalability parallelism**



Node scalability parallelism in Flowbster

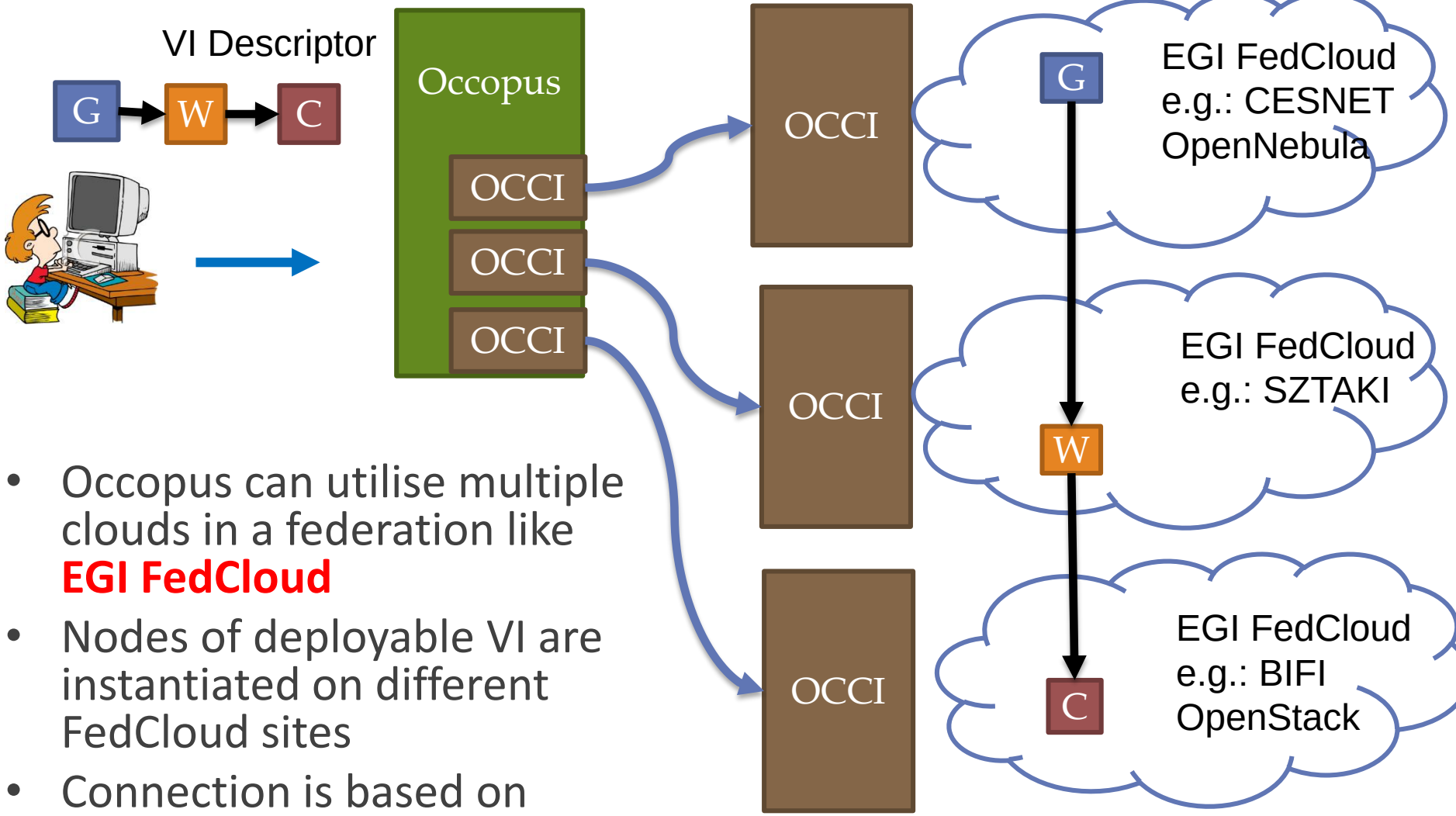
Generator-Worker-Collector parameter sweep processing pattern:

- The Generator generates N output data from 1 input data
- The Worker should be executed for every input data -> **N Worker instances can run in parallel** for processing the N data
- The Collector collects the N results coming from the N Worker instances and after processing them creates 1 output data



Heterogeneous multi-cloud setup of Flowbster in EGI FedCloud

MTA SZTAKI
Hungarian Academy of Sciences
Institute for Computer Science and Control



- Occopus can utilise multiple clouds in a federation like **EGI FedCloud**
- Nodes of deployable VI are instantiated on different FedCloud sites
- Connection is based on public ips

Performance measurements

- Experiment with Autodock Vina Workflow
- **Question:** What speedup can be achieved by node scalability parallelism?
- 256 docking simulation having 2, 4, 8 and 16 instances of the Vina (worker) node

Case (number of Vina node instances)	Makespan (minutes)
2	8
4	5
8	3
16	2

Current state of Occopus

- Open-source (License: Apache v2)
- 6 releases so far (latest in August 2016)
- **Now: Release v1.2 (3rd production release)**
- Python 2.7
- Base webpage: http://occopus.lpds.sztaki.hu
- Git: https://github.com/occopus
- Documentation:
  - Users' Guide
  - Developers' Guide
  - Tutorials (e.g. building docker/swarm cluster)
- Package repository: http://pip.lpds.sztaki.hu/packages

Current state of Flowbster

- Open-source (License: Apache v2)
- Running prototype
- **First release comes in October 2016**
- Available at Git: https://github.com/occopus
- Documentation under development:
  - Users' Guide
  - Developers' Guide
  - Tutorials
- Further development plans
  - Dynamic scalability for node scalability parallelism
  - Built-in error diagnostic and fault-recovery mechanism