1.简介

Apache Storm是由Twitter公司开源的一个分布式实时计算框架, 擅长进行实时数据流的处理, 并且可以和任何编程语言一起使用.
具有以下特点:
(1)易于与其他队列和数据库集成;

(2)易于使用: 只需定义3种抽象: spout, bolt, topology; 就可以实现处理和传递数据;

(3)易于扩展: 分布式运行, topo的每个部分都可以调整并发大小, 还有rebalance命令来动态调整并发量;

(4)较强的鲁棒性: 某个worker死掉会自动重启, 某个节点死掉该节点的worker会在另一个节点上重启;

(5)容错机制保证每条数据至少被处理一次;

(6)任何编程语言均可使用, 非java语言通过 "Multi-Language Protocol"与storm通信;

(7)易于部署和操作;

2.概念

(1)Nimbus: storm集群的master节点, 负责向各节点分发代码, 分配任务以及监控节点的运行状态;

(2)Supervisor: 每个工作节点运行一个supervisor进程, 负责接收nimbus指派的任务, 根据任务开启或停止worker进程, supervisor和nimbus通过zookeeper通信;

(3)Worker: 每个工作节点上具体执行数据处理逻辑的进程, 不同worker间通过Netty来通信;

(4)Topology: spout和bolt的连接图,规定数据的处理逻辑和传递路线, 不同机器上的多个worker组成topology;

(5)Executor: 每个worker下的1个线程称为executor, executor中执行一个或多个task;

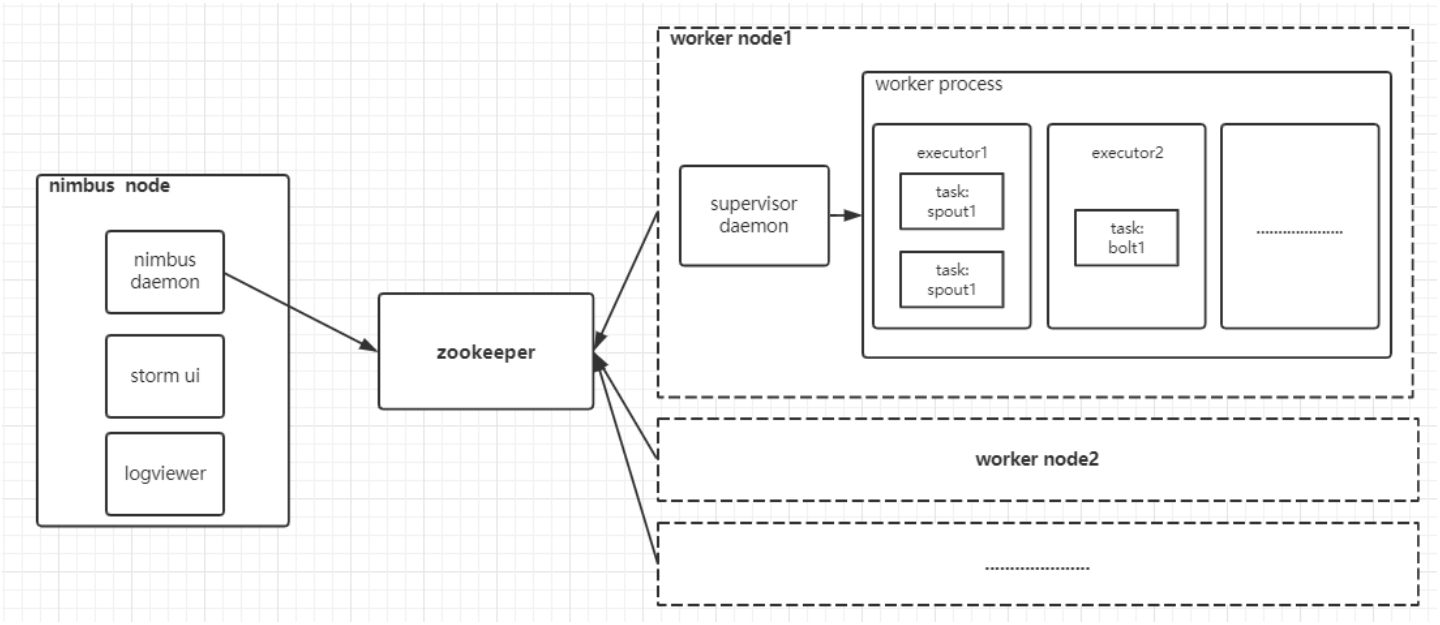
(6)Task: 每个spout和bolt都会根据各自的并发量设置被当做一个或多个task执行;

(7)Spout: 产生数据源的具体逻辑, 可以自己生成或从外部读取;

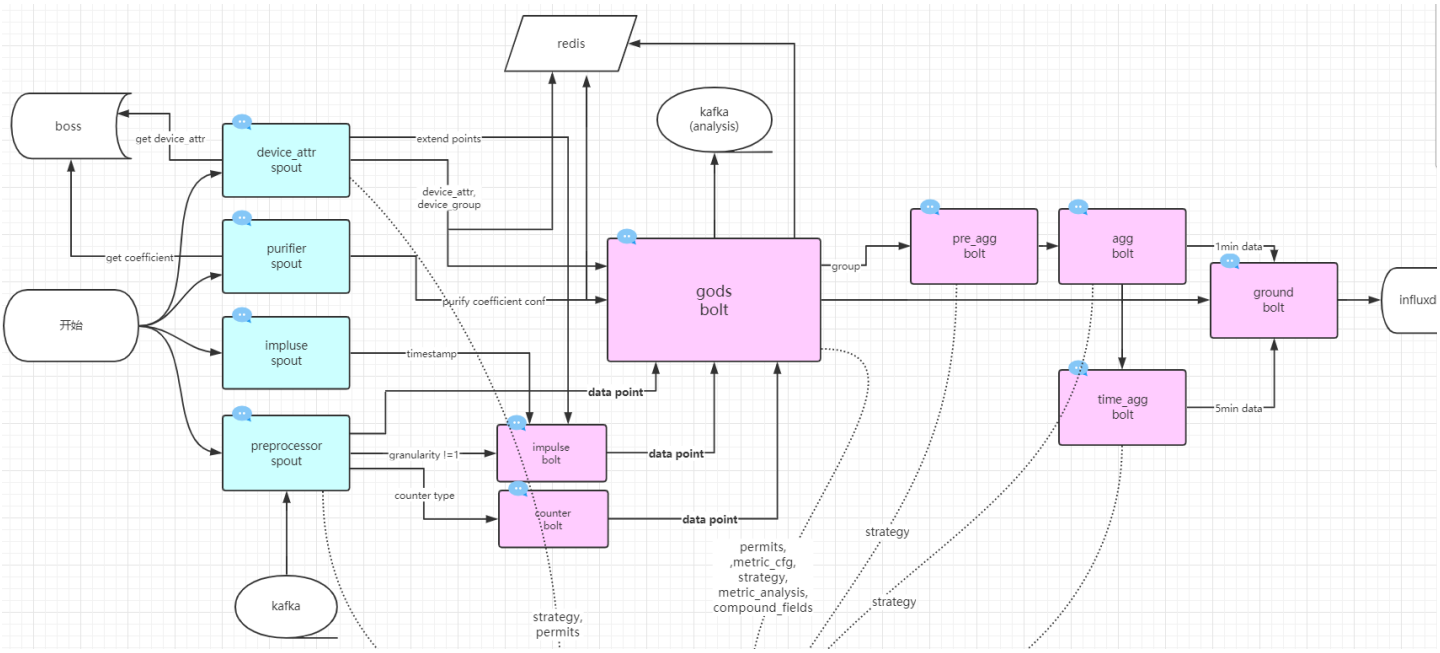
(8)Bolt: 处理, 计算数据的具体逻辑;

(9)Tuple: storm的数据模型, 数据流中的基本处理单元;

(10)Groupings: tuple在各task间的分发策略: Shuffle grouping, Fields grouping等;



3. pcdn实时流处理



4.streamparse

(1) project contents:

sparse quickstart project_name

File/Folder	Contents
config.json	Configuration information for all of your topologies.
fabfile.py	Optional custom fabric tasks.
project.clj	leiningen project file (can be used to add external JVM dependencies).
src/	Python source files (bolts/spouts/etc.) for topologies.
tasks.py	Optional custom invoke tasks.
topologies/	Contains topology definitions written using the Topology DSL.
virtualenvs/	Contains pip requirements files used to install dependencies on remote Storm servers.

(2) spouts:

```
import itertools
from streamparse.spout import Spout

class SentenceSpout(Spout):
    outputs = ['sentence']

    def initialize(self, stormconf, context):
        self.sentences = [
            "She advised him to take a long holiday, so he immediately quit work and took a tri
            "I was very glad to get a present from her",
            "He will be here in half an hour",
            "She saw him eating a sandwich",
        ]
        self.sentences = itertools.cycle(self.sentences)

    def next_tuple(self):
        sentence = next(self.sentences)
        self.emit([sentence])
        # emit(tup, tup_id=None, stream=None, direct_task=None, need_task_ids=False)

    def ack(self, tup_id):
        pass  # if a tuple is processed properly, do nothing

    def fail(self, tup_id):
        pass  # if a tuple fails to process, do nothing
```

(3) bolts:

```
import re
from streamparse.bolt import Bolt

class SentenceSplitterBolt(Bolt):
    outputs = ['word']
    auto_ack = True
    auto_fail = True
    auto_anchor = True

    def process(self, tup):
        sentence = tup.values[0]  # extract the sentence
        sentence = re.sub(r"[,.;!\?]", "", sentence)  # get rid of punctuation
        words = [[word.strip()] for word in sentence.split(" ") if word.strip()]
        if not words:
            # no words to process in the sentence, fail the tuple
            self.fail(tup)
            return

        for word in words:
            self.emit([word])
            # emit(tup, stream=None, anchors=None, direct_task=None, need_task_ids=False)
```

(4) topology:

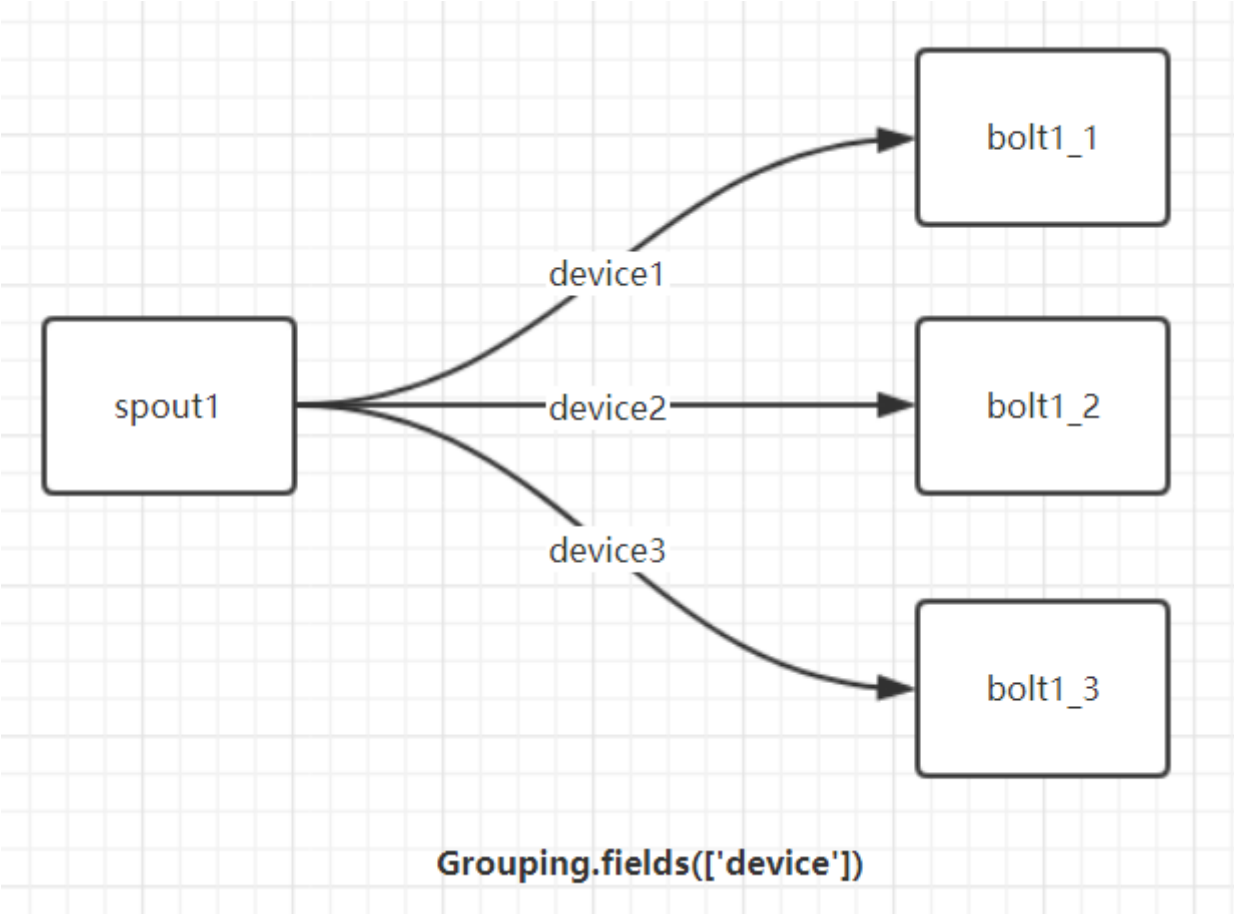
```
from streamparse import Grouping, Topology

from bolts.wordcount import WordCountBolt
from spouts.words import WordSpout


class WordCount(Topology):
    word_spout = WordSpout.spec()  # spec(name=None, inputs=None, par=None, config=None)
    count_bolt = WordCountBolt.spec(inputs={word_spout: Grouping.fields("word")}, par=2)
```

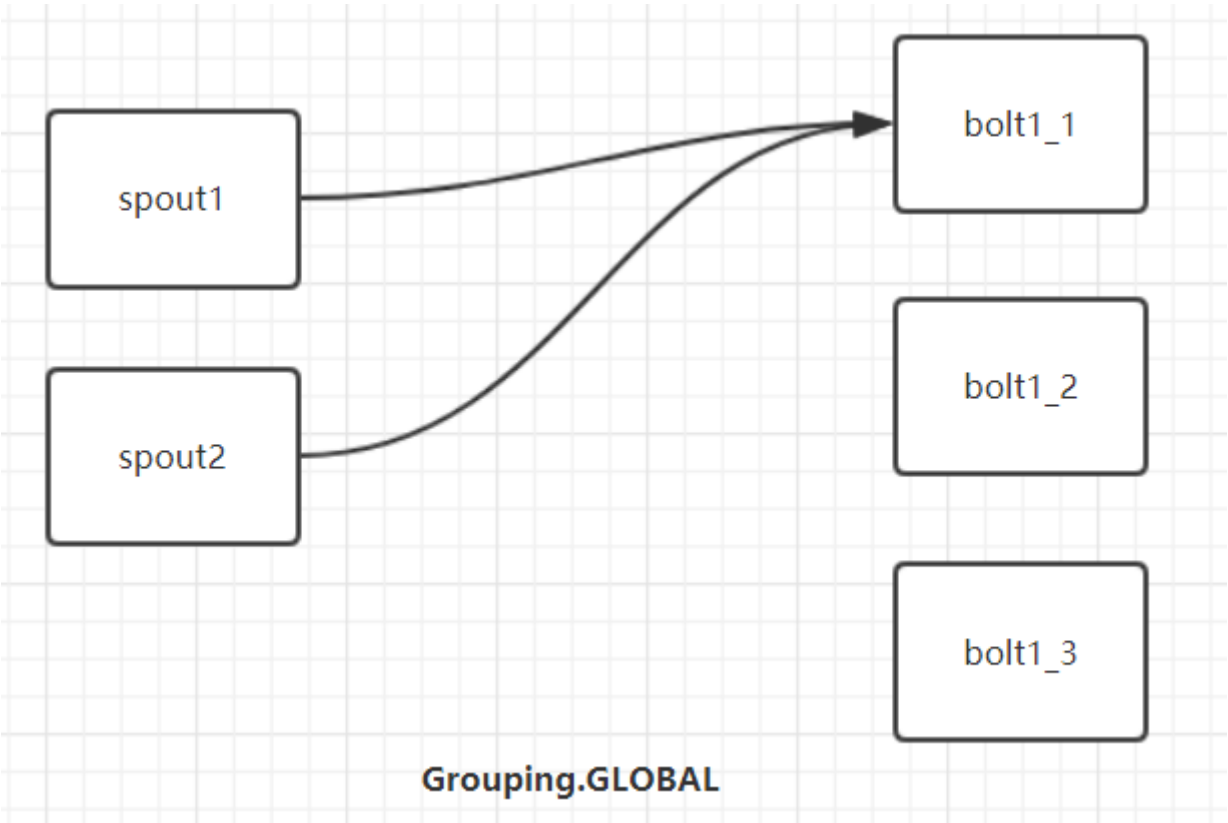
(5) grouping

field grouping

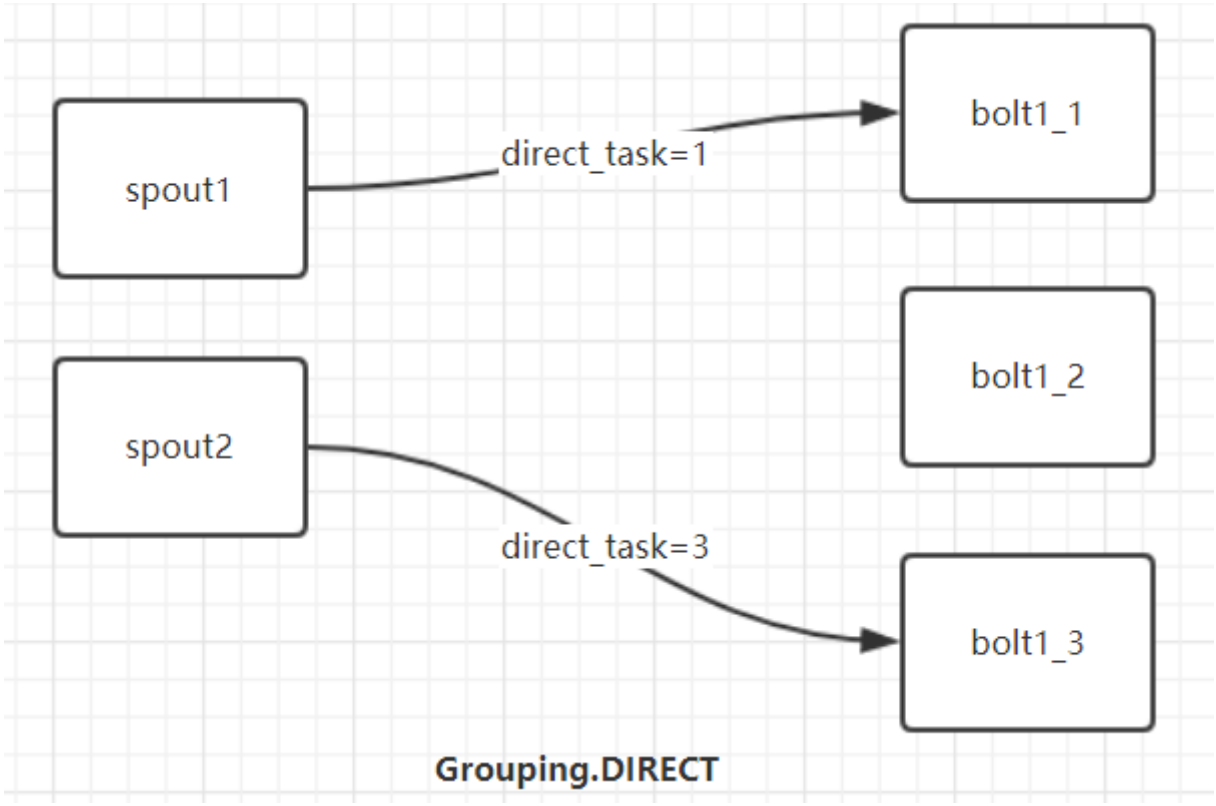


Shuffle grouping

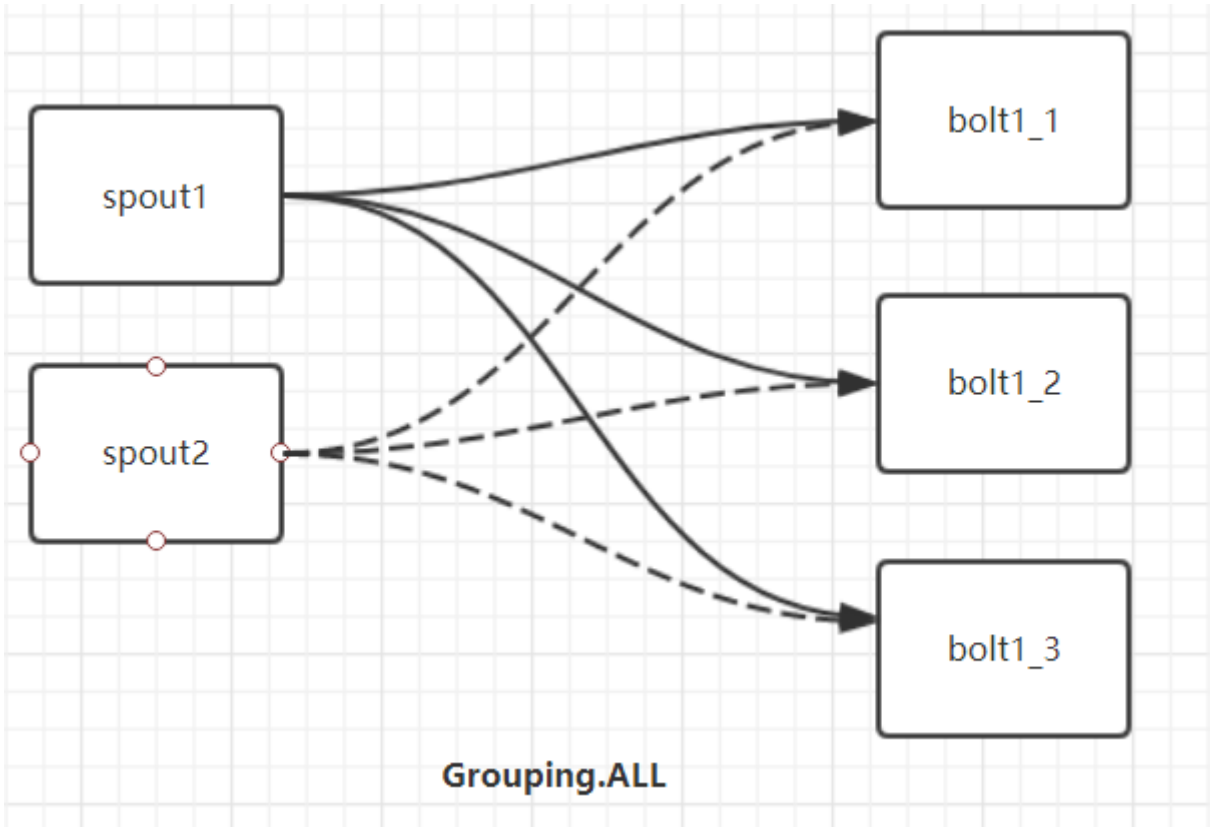
global grouping: 流向id最小的task



direct grouping: 指向某个task



all grouping: 流向所有task



none grouping: 相当于shuffle

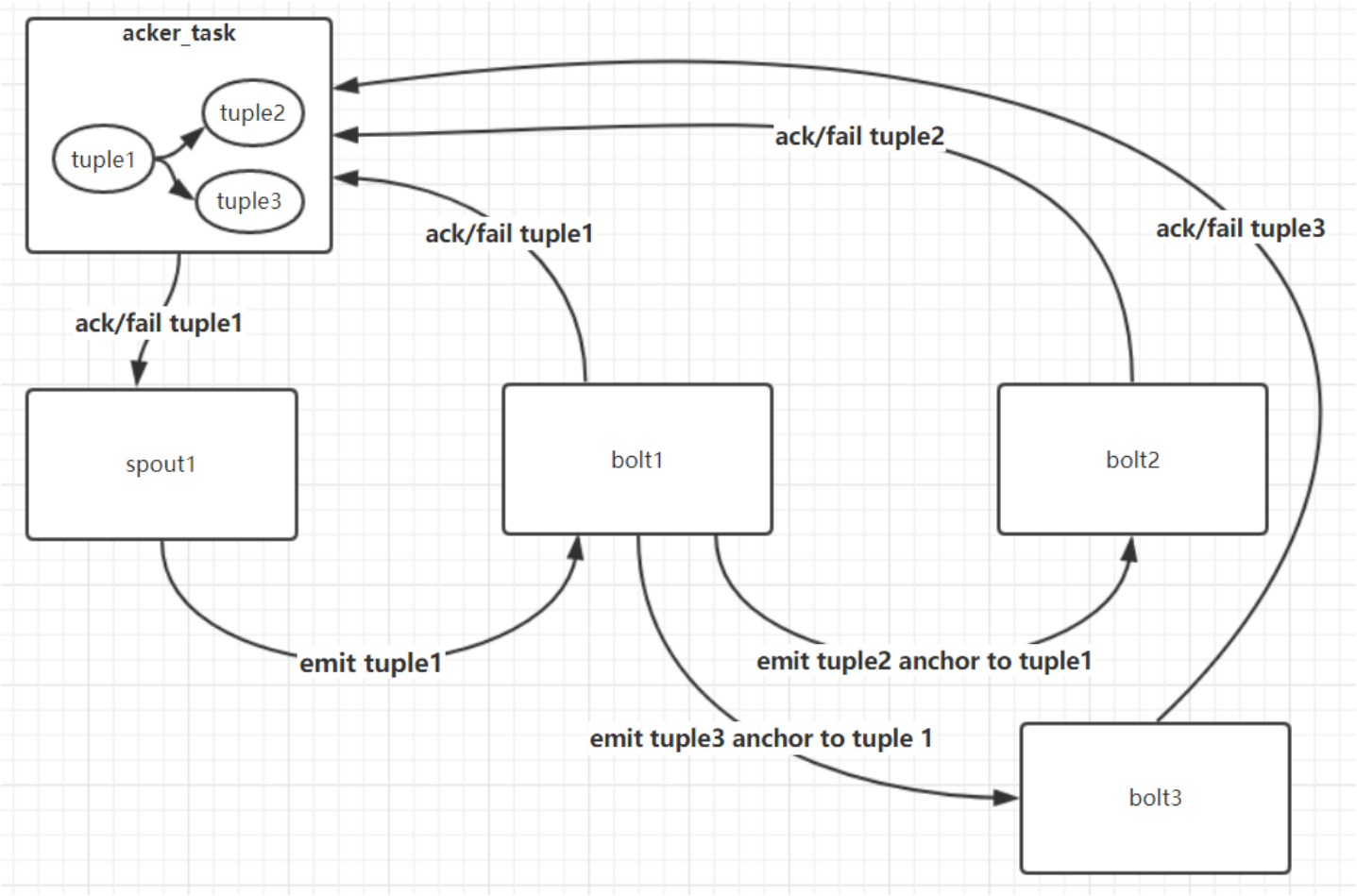
local_or_shuffle grouping: 随机但优先本地

(6) outout stream

```
outputs = [
        Stream(fields=["device", "point"], name='default'),
        Stream(fields=["type", "device", "point"], name='agg'),
]
```

(7) reliable:

(1)auto_ack, auto_fail和auto_anchor



(2)reliable spout和spout

(8) 使用注意:

next_tuple(), process()非阻塞;
程序中禁止出现print等占用标准输入输出等语句;

5.配置

级别: storm的配置主要分为系统级别和topology级别(以topology开头);

定义方式:

(1)conf/storm.yaml

(2)对于streamparse: 配置config.json或命令行;

常用配置项:
storm.zookeeper.servers
storm.local.dir
nimbus.seeds
supervisor.slots.ports

topology.max.task.parallelism: 每个component最大tasks数
topology.max.spout.pending: 每个spout中正在处理的最大tuple数
topology.debug: 是否以debug级别运行
topology.workers: 启动的worker数
topology.worker.childopts: 给相关Java worker传递参数

6.运行模式

local mode: sparse run

cluster mode: sparse submit

7.storm ui

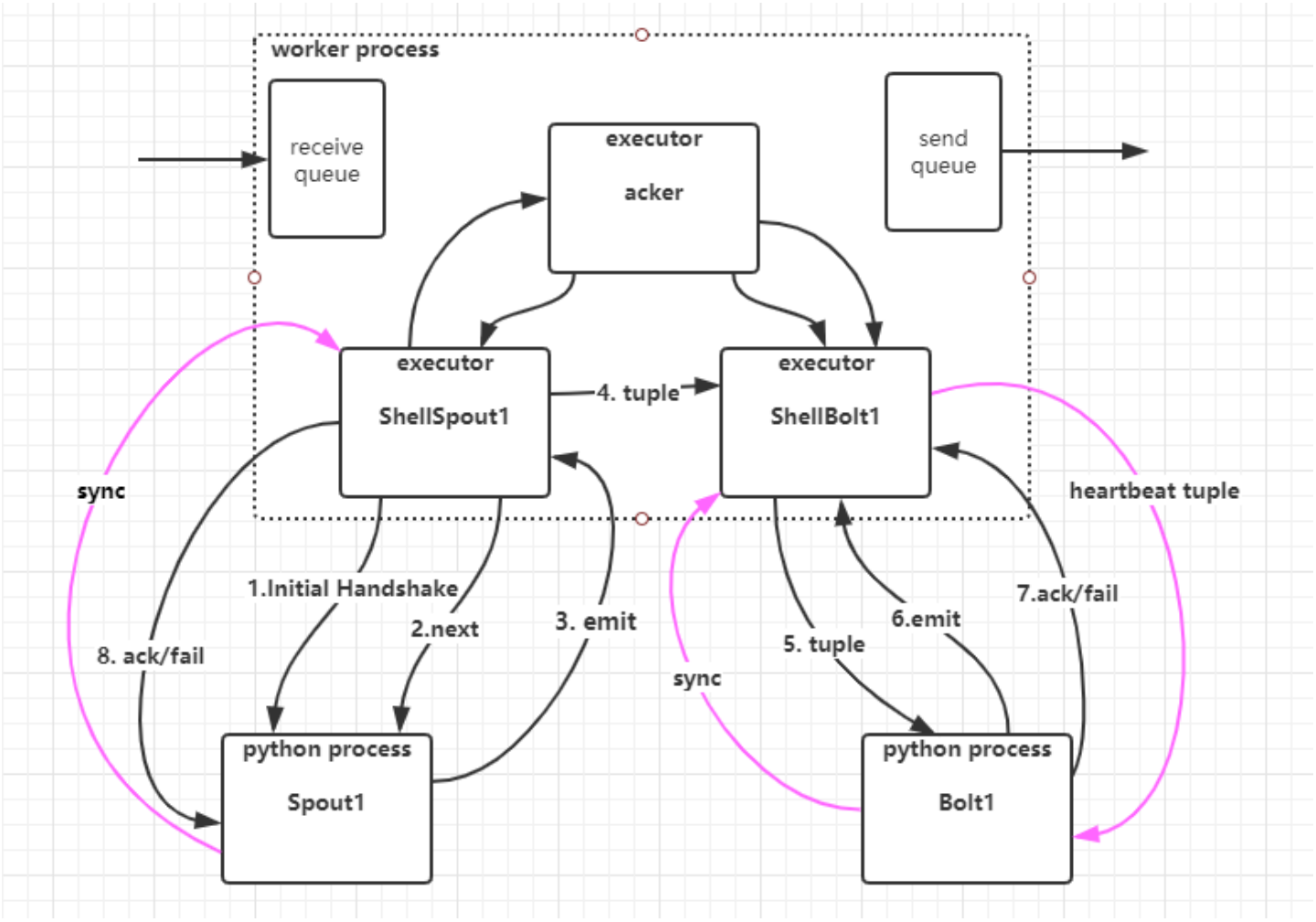
(1) 监控运行指标

(2) 操作storm

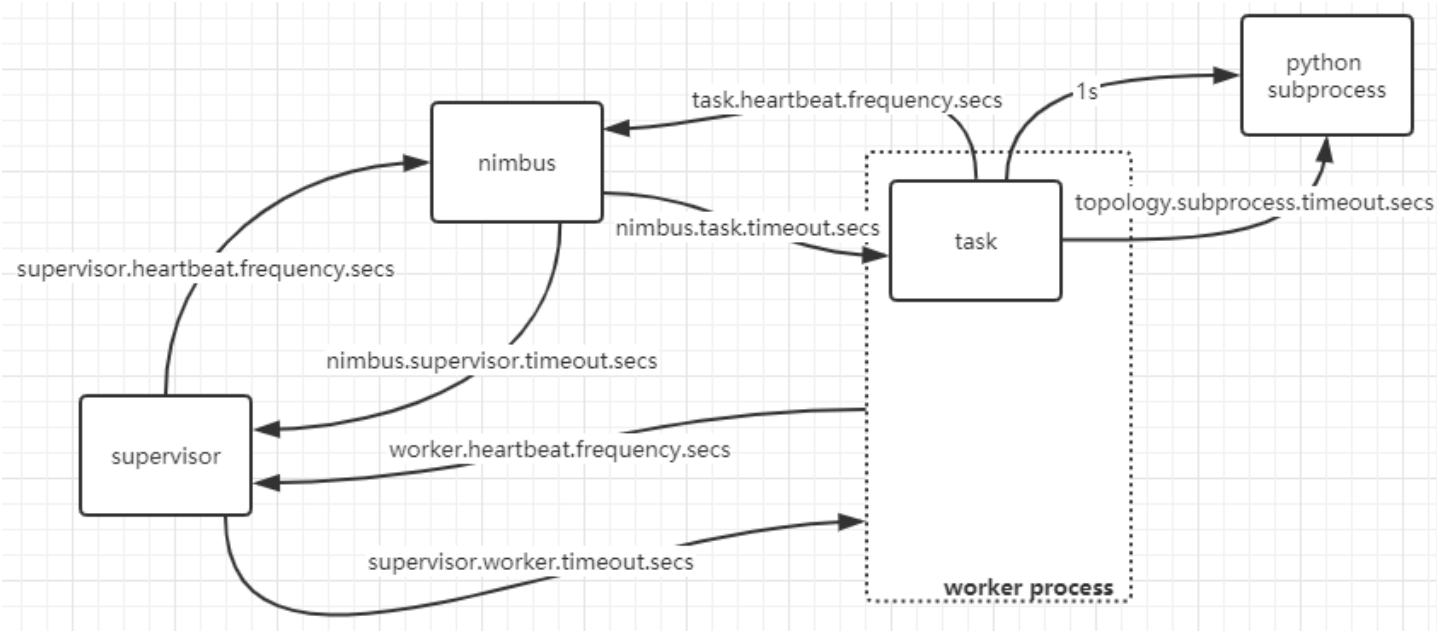
(3) 查看日志

8. Storm Multi-Language Protocol

python实现: streamparse



9. heartbeat



supervisor.worker.timeout.secs: 30
worker.heartbeat.frequency.secs: 1

supervisor.heartbeat.frequency.secs: 5
nimbus.supervisor.timeout.secs: 60

task.heartbeat.frequency.secs: 3
nimbus.task.timeout.secs: 30

topology.subprocess.timeout.secs