Bioinformatics, 2017, 1–3
doi: 10.1093/bioinformatics/btx403
Advance Access Publication Date: 19 June 2017
Applications Note

OXFORD

Genome analysis

BioQueue: a novel pipeline framework to accelerate bioinformatics analysis

Li Yao[1,*], Heming Wang[2], Yuanyuan Song[1] and Guangchao Sui[1,*]

[1]College of Life Science, Northeast Forestry University, Harbin 150040, China and [2]School of Life Science and Technology, Shanghai Tech University, Shanghai 200031, China

*To whom correspondence should be addressed.

Associate Editor: John Hancock

Received on March 27, 2017; revised on May 25, 2017; editorial decision on June 15, 2017; accepted on June 16, 2017

Abstract

Motivation: With the rapid development of Next-Generation Sequencing, a large amount of data is now available for bioinformatics research. Meanwhile, the presence of many pipeline frameworks makes it possible to analyse these data. However, these tools concentrate mainly on their syntax and design paradigms, and dispatch jobs based on users' experience about the resources needed by the execution of a certain step in a protocol. As a result, it is difficult for these tools to maximize the potential of computing resources, and avoid errors caused by overload, such as memory overflow.

Results: Here, we have developed BioQueue, a web-based framework that contains a checkpoint before each step to automatically estimate the system resources (CPU, memory and disk) needed by the step and then dispatch jobs accordingly. BioQueue possesses a shell command-like syntax instead of implementing a new script language, which means most biologists without computer programming background can access the efficient queue system with ease.

Availability and implementation: BioQueue is freely available at https://github.com/liyao001/BioQueue. The extensive documentation can be found at http://bioqueue.readthedocs.io.

Contact: li_yao@outlook.com or gcsui@nefu.edu.cn

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Using pipeline frameworks to analyze data produced by the Next-Generation Sequencing techniques is now a common task in current biomedical research. Basically, a pipeline framework provides two functions; the explicit one is the description of protocols (or workflow, pipe-line called in other software), and the ambiguous one is how to dispatch jobs to achieve a maximum efficiency in analyzing data.

Actually, many frameworks focus on a design philosophy of describing a protocol to make it compatible with various demands and easy to scale up from a single laptop to clouds or clusters. For example, Galaxy (Goecks *et al.*, 2010) provides pre-defined toolsheds and a web-based Graphical User Interface (GUI) to help users to create protocols. Since GUI may limit the flexibility, other queue systems emerged with command line interface and a build-in domain-specific language (DSL) to describe protocols, such as Ruffus (Goodstadt, 2010) and BigDataScript (Cingolani *et al.*, 2015). This improvement can ensure the flexibility, but it leads to a steep learning curve. On the other hand, it is common for pipelines to run multiple jobs in parallel to improve the efficiency of analysis. Importantly, all tools mentioned above assume that users have extensive knowledge in properly allocating system resources for each job when analyzing data, which obviously does not apply to many circumstances.

Here, we introduce BioQueue, a novel queue system that, when dispatching jobs, can automatically estimate the system resources needed by the execution of a certain step and then optimize the execution of the queue to maximize the efficiency. Additionally, based on a design paradigm of configuration, BioQueue provides a web-based workbench and implements an explicit syntax (Leipzig, 2016) to keep an optimal balance between flexibility and simplicity.

2 Job dispatch

The aim of BioQueue is to improve the efficiency and robustness of bioinformatics research. One common strategy is to use as many as CPU cores available to reduce time elapsed on context switching. The other strategy is to run different jobs in parallel to efficiently use computing resources. However, due to the design of a software or limited system resources, many currently available tools have following flaws:

cannot fully use the system resources allotted to them (see Supplementary Fig. S1).

may not achieve an ideal efficiency or can even generate errors (such as memory overflow) when running multiple jobs simultaneously.

To circumvent these problems, BioQueue will calculate the hash for a certain step and then check the presence of any existing prediction model for it. If a model of predicting the sources required by the step is generated, BioQueue will use it; otherwise, BioQueue will collect the size of inputs, CPU utilization, memory usage (Supplementary Note S1) and disk usage information of this step as training materials. Resources needed by the step will be estimated by the following function:

$$f(x) = \begin{cases} The\ mean\ of\ training\ data & |r| < threshold \\ ax + b & |r| \geq threshold \end{cases}$$

When running on a personal computer or clouds, BioQueue implements a greedy algorithm to dispatch jobs with these estimated data to achieve an optimal balance between the efficiency and system resources (CPU, memory and disk). One special circumstance is that, when running a step for the first time, BioQueue will strictly limit the parallelism to reduce the possibility of creating an error. We benchmarked BioQueue against other four popular pipeline frameworks following the protocol presented in Supplementary Figure S2 and found that BioQueue can significantly shorten the elapsed time (Fig. 1 and Supplementary Fig. S3).

We also developed a plug-in system (Supplementary Note S2) to provide support to various distributed resource managers (DRMs), so BioQueue can run on clusters smoothly. On these platforms, BioQueue cooperates with DRMs by guiding them to allocate a proper quantity of CPU cores and memory for the execution of each step. More detailed information regarding how BioQueue generates a prediction model is depicted in Supplementary Figures S4 and S5.

3 Language design

One very conspicuous characteristic of data analysis in bioinformatics is that researchers usually need to analyze a large amount of data by using a particular protocol and other researchers may also need to use the same protocol to analyze their own data. Therefore, improving the reusability of a protocol is very crucial. To achieve this goal, BioQueue explicitly differentiates two concepts. One concept is a 'protocol', a chain of steps consisting of software and its parameters that define the behavior of the analysis. The second concept is a 'job'. When a 'protocol' is assigned with specific experimental variables, like input files, output files or sample name, the protocol will turn into a runnable 'job'.

When creating a protocol, BioQueue implements a shell command-like syntax with wildcards (Table 1 shows a typical protocol in BioQueue) rather than a new script language. This is user-friendly to general biologists, because they do not have to learn programming, but only need to know the parameters of the tools that are easy to be found at the tools' papers. Though BioQueue explicitly separates the concepts of 'protocol' and 'job', a protocol can be converted into a runnable job without assigning any experimental values. However, we strongly recommend researchers to replace those values with wildcards (strings embraced with braces, like '{ThreadN}') to make the protocol reusable and reproducible. There are three types of wildcards in BioQueue:

- Pre-defined wildcards, which provide support for file mapping and multithread.
- References, the reference data which might be cited in multiple protocols, such as reference human genome and gene annotation.
- User-defined wildcards, or experimental variables, which need to be assigned when creating a job.

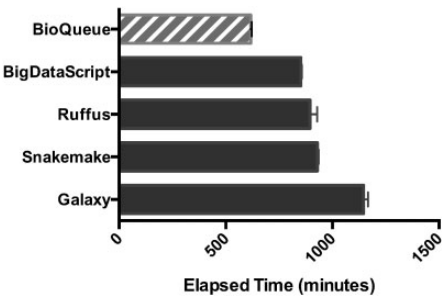

Fig. 1. Benchmark results of BioQueue against other four popular pipelines using a previously published RNA-seq dataset (E-MTAB-513). Error bars = SD ($n = 3$)

4 Auxiliary functions

To facilitate biologists to use BioQueue, we provide auxiliary instructions that cover the entire process from writing a protocol to creating a job.

Firstly, we have made the protocols in BioQueue transplantable, which allows users who are using different instances to import suitable protocols created by others and process the same analysis. As a result, any BioQueue user can export a protocol with a generated model as a plain text file, and then upload the protocol to any forum or our open platform (http://bioqueue.nefu.edu.cn).

Secondly, BioQueue offers an open knowledge base to all biologists. On BioQueue's workbench, users can not only search for the usages of bioinformatics software suitable to their own analysis, but also contribute to this knowledge base by sharing new usage information, which will be beneficial to subsequent users. In this manner, the knowledge base will be vigorously maintained by the community with the most updated software and protocols.

Thirdly, we have embedded an autocomplete widget to provide suggestions among pre-defined wildcards and references when users type parameters in each step.

Lastly, when creating a new job, an auto-detecting widget can remind the user of the wildcards defined in a selected protocol.

Table 1. A typical protocol (Pertea *et al.*, 2016) implemented in BioQueue

Step	Software	Parameter
1	hisat2	−p {ThreadN} –dta -x {HISAT2_HG38} -1 {Input-File:1} -2 {InputFile:2} -S {EXP}.sam
2	Samtools	sort -@ {ThreadN} -o {EXP}.sorted.bam {EXP}.sam
3	stringtie	−p {ThreadN} -G {GENCODE_HG38_V23} -o {EXP}.gtf -l {EXP} {EXP}.sorted.bam

Funding

This work was supported by the National Nature Science Foundation of China [81472635 and 81672795] to G.S.

Conflict of Interest: none declared.

References

Cingolani,P. *et al*. (2015) BigDataScript: a scripting language for data pipelines. *Bioinformatics*, **31**, 10–16.

Goecks,J. *et al*. (2010) Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*., **11**, R86.

Goodstadt,L. (2010) Ruffus: A lightweight python library for computational pipelines. *Bioinformatics*, **26**, 2778–2779.

Leipzig,J. (2016) A review of bioinformatic pipeline frameworks. *Brief. Bioinform*., bbw020.

Pertea,M. *et al*. (2016) Transcript-level expression analysis of RNA-seq experiments with HISAT, StringTie and Ballgown. *Nat. Protoc*., **11**, 1650–1667.