Building Applications for Interactive Data Exploration in Systems Biology

Bjørn Fjukstad
Department of Computer Science
UiT The Arctic University of Norway

Vanessa Dumeaux
Department of Biology
Concordia University

Karina Standahl Olsen
Department of Community Medicine
UiT The Arctic University of Norway

Eiliv Lund
Department of Community Medicine
UiT The Arctic University of Norway

Michael Hallett
Department of Biology
Concordia University

Lars Ailo Bongo
Department of Computer Science
UiT The Arctic University of Norway

ABSTRACT

The significant increase in the rate of data generation by the systems biology community creates a need for interactive exploration tools to explore the resultant datasets. Such tools need to combine advanced statistical analyses, prior knowledge from biological databases, and interactive visualizations with intuitive user interfaces. Each specific research question potentially requires a specialized user interface and visualization methods. Although some features are application-specific, the underlying components of the data analysis tool can be shared and reused.

Our approach for developing data exploration tools in systems biology builds on the microservice architecture that separates an application into smaller components which can communicate using language-agnostic protocols. We show that this design is well suited for bioinformatics applications where different tools written in different languages by different research groups is the norm. Packaging each service in a software container enables re-use and sharing of key components between applications, reducing development, deployment, and maintenance time.

We demonstrate the viability of our approach through a web application, entitled MIxT blood-tumor, for exploring and comparing transcriptional profiles from blood and tumor samples in breast cancer patients. The application integrates advanced statistical software, up-to-date information from biological databases, and modern data visualization libraries.

ACM-BCB'17, August 20-23, 2017, Boston, MA, USA.
 978-1-4503-4722-8/17/08. . . $15.00
DOI: http://dx.doi.org/10.1145/3107411.3107481

KEYWORDS

Interactive data exploration, software containers, visualization, microservices, systems biology, breast cancer.

INTRODUCTION

In recent years the biological community has generated an unprecedented amount of data. While the cost of data collection has drastically decreased, data analysis continue to represent a large fraction of the total cost of these studies.[9] Data analysis tools, especially those designed specifically for the project at hand, provide clear benefit to the human experts who are interpreting data and deriving results.

Often in systems biology studies, the ability to explore newly generated data relative to prior knowledge located in third-party databases and software systems is key. This includes, for example, entities such as the Gene Ontology (GO),[1] the Kyoto Encyclopedia of Genes and Genomes (KEGG),[2] and the Molecular Signatures Database (MSigDB)[3] that together catalog the function of nearly every gene, gene product, pathway or cellular process. These tools, and most bioinformatics databases in general, offer interfaces for data retrieval.

Data analysis in systems biology is greatly reliant on programming languages especially tailored to these domains, providing easy direct access to specific algorithms and statistical routines. The R statistical programming language provides developers open access to thousands of libraries through repositories such as CRAN[4] or Bioconductor[5]. Similarly, other languages such as Python and Go have bioinformatic extensions including BioPython[2] and biogo[6] respectively, providing domain specific routines. Although tremendously helpful, different tools and languages are used in different domains of systems biology for many reasons. This creates a

[1] geneontology.org.
[2] kegg.jp.
[3] software.broadinstitute.org/gsea/msigdb.
[4] cran.r-project.org.
[5] bioconductor.org.

need for novel approaches to integrate the different libraries between the programming languages and tools.

A microservice architecture structures an application into small reusable, loosely coupled parts. These communicate via lightweight programming language-agnostic protocols such as HTTP, thus making it possible to write single applications in multiple programming languages. This way the most suitable programming language is used for each specific part. To build a microservice application, developers bundle each service in a software container. Containers are built from configuration files which describe the operating system, software packages and their associated versions. Several software container implementations exist including Rkt[6] but Docker,[7] is perhaps the most broadly used. Initiatives such as BioContainers[8] now provide containers pre-installed with different bioinformatics tools. While the enabling technology is available, the microservices approach is not yet widely adopted in bioinformatics.[11]

From our experience we identified a set of components and features that are central to building data exploration applications.

(1) A low-latency language-independent approach for integrating, or embedding, statistical software, such as R, directly into a data exploration application.
(2) Low latency language-independent interface to online reference databases in biology that users can query to explore analyses.
(3) A simple method for deploying and sharing the components of an application between projects.

In this paper, we describe a novel approach for building data exploration applications in systems biology via a sample web application, MIxT (Matched Interactions across Tissues) using high-throughput gene expression profiles of breast cancer tumor data with matched profiles from the patients blood.

METHODS

In this section we first motivate our microservice approach based on our experiences developing the MIxT web application. We describe the process from initial data analysis to the final application, highlighting the importance of language-agnostic services to facilitate the use of different tools in different parts of the application. We then generalize the ideas to a set of principles and services that can be reused and shared between applications, and show their design and implementation.

Motivating Example

The aim of the Matched Interactions Across Tissues (MIxT) study was to identify genes and pathways in the primary breast tumor that are tightly linked to genes and pathways in the patient blood cells.[3] We generated and analyzed expression profiles from blood and matched tumor cells in 173 breast cancer patients included in the Norwegian Women and Cancer (NOWAC) study. The MIxT analysis starts by identifying sets of genes tightly co-expressed across all patients in each tissue. Each group of genes or modules were annotated based on a priori biological knowledge about gene functionality. Focus was placed on the relationships between tissues by asking if specific biologies in one tissue are linked with (possibly distinct) biologies in the second tissue, and this within different subgroup of patients (i.e. subtypes of breast cancer).

We built an R package, mixtR,[9] with the statistical methods and static visualizations for identifying associations between modules across tissues. To make the results more easily accessible we built a web application that interfaces with the R package, but also online databases to retrieve relevant metadata. To make it possible to easily update or re-implement parts of the system without effecting the entire application, it was developed using a microservice architecture. The software containers allowed the application to be deployed on a wide range of hardware, from local installations to cloud systems.

Design Principles

Our experience can be generalized into the following design principles for building applications in bioinformatics:

Principle 1: Build applications as collections of language-agnostic microservices. This enables re-use of components and does not enforce any specific programming language on the user interfaces or the underlying components of the application.

Principle 2: Use software containers to package each service. This has a number of benefits: it simplifies deployment, ensures that dependencies and libraries are installed, and simplifies sharing of services between developers.

Microservice Design and Implementation

In the rest of the section we describe how we designed and implemented two microservices in Kvik[4] which we later used to build the MIxT web application.

Compute Service. The main goal of a data exploration application in systems biology is to help users discover interesting patterns in a biological dataset. Because of the complexity of biological data and analyses, we need specialized software to help find these patterns. Because these tools are built to provide specialized analyses, they often don't provide a reusable interface outside the programming environment they are built in.

We have built a compute service that provides an open interface directly to the R programming language, thus providing access to a wealth of algorithm and statistical analysis packages that exists within the R ecosystem. Application developers can use the compute service to execute specialized analyses and retrieve results either as plain text or binary data such as plots. By interfacing directly with R, developers

[6] coreos.com/rkt.
[7] docker.com.
[8] biocontainers.pro.
[9] Available online at github.com/vdumeaux/mixtR.

can modify input parameters to statistical methods directly from the user-facing application.

The compute service offers three main operations to interface with R: i) to call a function with one or more input parameters from an R package, ii) to get the results from a previous function call, and iii) a catch-all term that both calls a function and returns the results. We use the same terminology as OpenCPU[8] and have named the three operations Call, Get, and RPC respectively. These three operations provide the necessary interface for applications to include statistical analyses in the applications.

The compute service is implemented as an HTTP server that communicates with a pre-set number of R processes to execute statistical analyses. At initiation of the compute service, a user-defined number of R worker sessions are launched for executing analyses (default is 5). The compute service uses a round-robin scheduling scheme to distribute incoming requests to the workers. We provide a simple FIFO queue for queuing of requests. The compute service also provides the opportunity for applications to cache analysis results to speed up subsequent calls.

Database Service. To interpret data, experts regularly exploit prior knowledge via database queries and the primary scientific literature. There are a wealth of online databases, some of which provide open APIs in addition to web user interfaces that application developers can make use of. While the databases can provide helpful information, there are some limitations associated with their integration into interactive data exploration applications: i) the APIs are not fast enough to use in interactive applications where the application has to perform multiple database queries, ii) some databases put restrictions on the number of database queries, and iii) there is no uniform way for storing additional database metadata to identify database versions and query parameters.

To alleviate application developers of these challenges, we built an database service that provides a solution to the three. The service provides low latency, minimizes the number of queries to remote databases, and stores additional metadata to capture query parameters and database information. The database service provides an open HTTP interface to biological databases for retrieving meta-data on genes and processes. We currently have packages for interfacing with E-utilities,[10] MSigDB, HGNC, and KEGG.

Both the compute and the databases service in Kvik build on the standard *net/http* package in the Go programming language.[11] The database service use the *gocache*[12] package to cache any query to an online database. In addition we deploy each service as Docker containers.[13]

[10] eutils.ncbi.nlm.nih.gov.
[11] golang.org
[12] github.com/fjukstad/gocache.
[13] Available at hub.docker.com/r/fjukstad/kvik-r and hub.docker.com/r/fjukstad/db.

MATCHED INTERACTIONS ACROSS TISSUES (MIXT)

We show the viability of the microservices approach in Kvik by describing the MIxT web application for exploring and comparing transcriptional profiles from blood and tumor samples. We conduct an initial evaluation to illustrate that we can built interactive applications using the microservices provided by Kvik.

Analysis Tasks

The web application provides functionality to perform six data analysis tasks (A1-A6):

A1: Explore co-expression gene sets in tumor and blood tissue. Users can explore gene expression patterns together with clinicopathological variables (e.g. patient or tumor grade, stage, age) for each module. In addition we enable users to study the underlying biological functions of each module by including gene set analyses between the module genes and known gene sets.

A2: Explore co-expression relationships between genes. Users can explore the co-expression relationship as a graph visualization. Here genes are represented in the network with nodes and edges represent statistically significant correlation in expression between the two end-points.

A3: Explore relationships between modules from each tissue. We provide two different metrics to compare modules, and the web application enables users to interactively browse these relationships. In addition to providing visualizations the compare modules from each tissue, users can explore the relationships, but for different breast cancer patient groups.

A4: Explore relationships between clinical variables and modules. In addition to comparing the association between modules from both tissues, users also have the possibility to explore the association with a module and a specific clinical variable. It is also possible to explore the associations after first stratifying the tumors by breast cancer subtype (an operation that is common in cancer related studies to deal with molecular heterogeneity).

A5: Explore association between user-submitted gene lists and computed modules. We want to enable users to explore their own gene lists to explore them in context of the co-expression gene sets. The web application must handle uploads of gene lists and compute association between the gene list and the MIxT modules on demand.

A6: Search for genes or gene lists of interest. To facilitate faster lookup of genes and biological processes, the web application provides a search functionality that lets users locate genes or gene lists and show association to the co-expression gene sets.

Design and Implementation

From these six analysis tasks we designed and implemented MIxT as a web application that integrates statistical analyses and information from biological databases together with interactive visualizations. Figure 1 shows the system architecture of MIxT which consists of three parts i) the web application

itself containing the user-interface and visualizations; ii) the compute service performing the MIxT analyses developed in an R package, delivering data to the web application; and iii) the database service providing up-to-date information from biological databases. Each of these components run within Docker containers making the process of deploying the application simple.

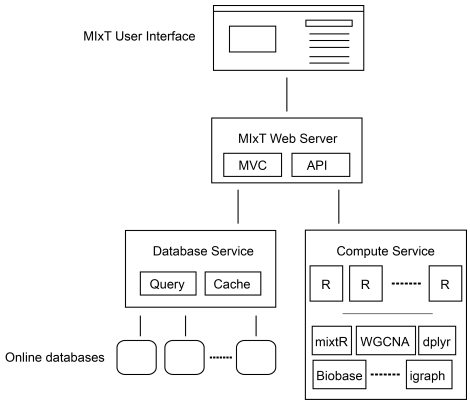

Figure 1: The architecture of the MIxT system. It consists of a web application, the hosting web server, a database service for retrieving metadata and a compute service for performing statistical analysis. Note that only the web application and the R package are specific to MIxT, the rest of the components can be reused in other applications.

We structured the MIxT application with a separate view for each analysis task. To explore the co-expression gene sets (**A1**), we built a view that combines both static visualizations from R together with interactive tables for gene overlap analyses. Figure 2 shows the web page presented to users when they access the co-expression gene set 'darkturquoise' from blood. To explore the co-expression relationship between genes (**A2**) we use an interactive graph visualization build with Sigmajs[14]. We have built visualization for both tissues, with graph sizes of 2705 nodes and 90 348 edges for the blood network, and 2066 nodes and 50 563 edges for the biopsy network. To visualize relationships between modules from different tissues (**A3**), or their relationship to clinical variables (**A4**) we built a heatmap visualization using the d3[15] library. We built a simple upload page where users can specify their gene sets (**A5**). The file is uploaded to the web application which redirects it to the compute service that runs the analyses. Similarly we can take user input to search for genes and processes (**A6**).

The web application is hosted by a custom web server. This web server is responsible for dynamically generating the different views based on data from the statistical analyses and biological databases, and serve these to users. It also

[14]sigmajs.org.
[15]d3js.org.

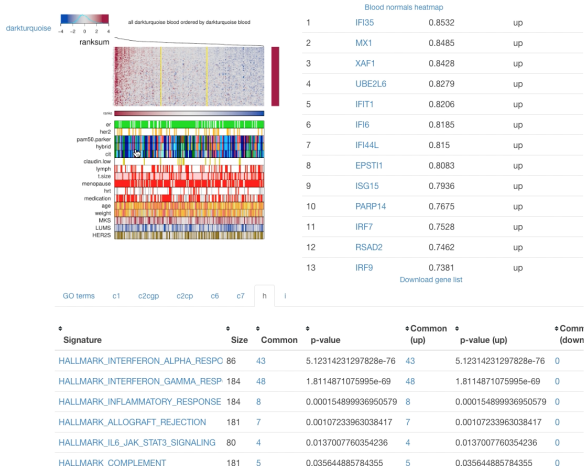

Figure 2: MIxT module overview page. The top left panel contains the gene expression heatmap for the module genes. The top right panel contains a table of the genes found in the module. The bottom panel contains the results of gene overlap analyses from the module genes and known gene sets from MSigDB.

serves the different JavaScript visualization libraries and style sheets.

Evaluation

To investigate if it is feasible to implement parts of an application as separate services, we evaluate the response times for a set of queries to each of the two supporting services.

To evaluate the database service we measure the query time for retrieving information about a specific gene with and without caching.[16] This illustrates how we can improve performance in an application by using a database service rather than accessing the database directly. We use a AWS EC2 *t2.micro*[17] instance to host and evaluate the database service. The results in Table 1 confirm a significant improvement in response time when the database service caches the results from the database lookups. In addition by serving the results out of cache we reduce the number of queries to the online database down to one.

Table 1: Time to retrieve a gene summary for a single gene, comparing different number of concurrent requests.

	1	2	5	10	15
No cache	956ms	1123ms	1499ms	2147ms	2958ms
Cache	64ms	64ms	130ms	137ms	154ms

We evaluate the compute service by running a benchmark consisting of two operations: first generate a set of 100

[16]More details online at github.com/fjukstad/kvik/tree/master/db/benchmark.
[17]See aws.amazon.com/ec2/instance-types for more information about AWS EC2 instance types.

random numbers, then plot them and return the resulting visualization.[18] We use two *c4.large* instances on AWS EC2 running the Kvik compute service and OpenCPU base docker containers. The servers have caching disabled. Table 2 shows the time to complete the benchmark for different number of concurrent connections. We see that the compute service in Kvik performs better than the OpenCPU[19] alternative. We believe that speedup is because we keep a pool of R processes that handle requests. In OpenCPU a new R process is forked upon every request that results in any computation executed in R. Other requests such as retrieving previous results do not fork new R processes.

Table 2: Time to complete the benchmark with different number of concurrent connections.

	1	2	5	10	15
Kvik	274ms	278ms	352ms	374ms	390ms
OpenCPU	500ms	635ms	984ms	1876ms	2700ms

RELATED WORK

In this section we discuss different methods that facilitates building applications using a microservices approach.

Integrate Statistical Analyses

OpenCPU is a system for embedded scientific computing and reproducible research.[8] Similar to the compute service in Kvik, it offers an HTTP API to the R programming language to provide an interface with statistical methods. It allows users to make function calls to any R package and retrieve the results in a wide variety of formats such as JSON or PDF. OpenCPU provides a JavaScript library for interfacing with R, as well as Docker containers for easy installation, and has been used to build multiple applications.[20]. The compute service in Kvik follows many of the design patterns in OpenCPU. Both systems interface with R packages using a hybrid state pattern over HTTP. Both systems provide the same interface to execute analyses and retrieve results. Because of the similarities in the interface to R in Kvik we provide packages for interfacing with our own R server or OpenCPU R servers.

Shiny is a web application framework for R[21] It allows developers to build web applications in R without having to have any knowledge about HTML, CSS, or Javascript. While it provides an easy alternative to build web applications on top of R, it cannot be used as a service in an application that implements the user-interface outside R.

Renjin is a JVM-based interpreter for the R programming language.[1] It allows developers to write applications in Java that interact directly with R code. This makes it possible to use Renjin to build a service for running statistical analyses on top of R. One serious drawback is that existing R packages must be re-built specifically for use in Renjin.

Visualization

Cytoscape is an open source software platform for visualizing complex networks and integrating these with any type of attribute data.[10] Through a Cytoscape App, cyREST, it allows external network creation and analysis through a REST API[7], making it possible to use Cytoscape as a service. To bring the visualization and analysis capabilities to the web applications the creators of Cytoscape have developed Cytoscape.js[22], a JavaScript library to create interactive graph visualizations. Another alternative for biological data visualization in the web browser is BioJS It provides a community-driven online repository with a wide range components for visualizing biological data contributed by the bioinformatics community.[5] BioJS builds on node.js[23] providing both server-side and client-side libraries. In MIxT we have opted to build the visualizations from scratch using sigma.js and d3 to have full control over the appearance and functionality of the visualizations.

Kvik and Kvik Pathwys

We have previously built a system for interactively exploring gene expression data in context of biological pathways.[4] Kvik Pathways is a web application that integrates gene expression data from the Norwegian Women and Cancer (NOWAC) cohort together with pathway images from the Kyoto Encyclopedia of Genes and Genomes (KEGG). We used the experience building Kvik Pathways to completely redesign and re-implement the R interface in Kvik. From having an R server that can run a set of functions from an R script, it now has a clean interface to call any function from any R package, not just retrieving data as a text string but in a wide range of formats. We also re-built the database interface, which is now a separate service. This makes it possible to leverage its caching capabilities to improve latency. This transformed the application from being a single monolithic application into a system that consists of a web application for visualizing biological pathways, a database service to retrieve pathway images and other metadata, and a compute service for interfacing with the gene expression data in the NOWAC cohort. We could then re-use the database and the compute service in the MIxT application.

DISCUSSION

There are different arguments for reusing and sharing microservices over libraries in bioinformatics applications, that would justify the cost of hosting an maintaining a set of distributed microservices. We argue that applications that require large computational or storage resources can benefit from the microservices approach because the applications can share the underlying compute infrastructure between

[18] More details at github.com/fjukstad/kvik/tree/master/r/benchmarks.
[19] Built using the *opencpu-server* Docker image.
[20] opencpu.org/apps.html.
[21] shiny.rstudio.com.
[22] js.cytoscapejs.org.
[23] nodejs.org.

multiple applications and users. This makes it possible to deploy an application on a lightweight system that uses a common service for computation and storage. In addition, benefits such as using different programming languages for a single application, and packaging a microservice as a software container, help to outweigh the operational burden related to using microservices to build applications.

We have used this approach to build different web applications and command line tools, but out of space constraints we only showcase one application in this paper. We have reused the microservices for running statistical analyses and fetch biological metadata, and share these between applications. This makes it possible for multiple applications to use one or more powerful servers for hosting the services. In the case of statistical analyses we simply install the necessary R packages for each application on the compute service and run it as we would for one single application.

Future work

We intend to address few points we aim to address in future work, both in the MIxT web application as well as the supporting microservices. The first issue is to improve the user experience in the MIxT web application. Since it is executing many of the analyses on demand, the user interface may seem unresponsive. We are working on mechanisms that gives the user feedback when the computations are taking a long time, but also reducing analysis time by optimizing the underlying R package. The database service provides a sufficient interface for the MIxT web application. While we have developed the software packages for interfacing with more databases, these haven't been included in the database service yet. In future versions we aim to make the database service an interface for all our applications. We also aim to improve how we capture data provenance. We aim to provide database versions and meta-data about when a specific item was retrieved from the database. One large concern that we haven't addressed in this paper is security. In particular one security concern that we aim to address in Kvik is the restrictions on the execution of code in the compute service. We aim to address this in the next version of the compute service, using methods such as AppArmor[24] that can restrict a program's resource access. In addition to code security we will address data access, specifically put constraints on who can access data from the compute service. We also aim to explore different alternatives for scaling up the compute service. Since we already interface with R we can use the Sparklyr[25] or SparkR[26] packages to run analyses on top of Spark.[12] Using Spark as an execution engine for data analyses will enable applications to explore even larger datasets.

CONCLUSIONS

We have designed an approach for building data exploration applications in systems biology that is based on a microservice architecture. Using this approach we have built a web application that leverages this architecture to integrate statistical analyses, interactive visualizations, and data from biological databases. While we have used our approach to build an application in systems biology, we believe that the microservice architecture can be used to build data exploration systems in other disciplines as well.

[24] wiki.ubuntu.com/AppArmor.
[25] spark.rstudio.com.
[26] spark.apache.org/docs/latest/sparkr.html.

ACKNOWLEDGMENTS

We would like to thank Andrew Bogecho and the System Staff at the School of Computer Science at McGill University for maintaining the compute infrastructure used to run the MIxT system.

This work has been funded by The European Research Council (ERC-AdG 232997 TICE), and The Canadian Cancer Society Research Institute (INNOV2-2014-702940).

REFERENCES

[1] Alexander Bertram. 2013. Renjin: The new R interpreter built on the JVM. In *The R User Conference, useR! 2013 July 10-12 2013 University of Castilla-La Mancha, Albacete, Spain*, Vol. 10. 105.

[2] Peter JA Cock, Tiago Antao, Jeffrey T Chang, Brad A Chapman, Cymon J Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and others. 2009. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* 25, 11 (2009), 1422–1423.

[3] Vanessa Dumeaux, Bjørn Fjukstad, Hans Fjosne E, Jan-Ole Frantzen, Marit Muri Holmen, Enno Rodegerdts, Ellen Schlichting, Anne-Lise Børresen-Dale, Lars Ailo Bongo, Eiliv Lund, and Michael T. Hallett. 2017. Interactions between the tumor and the blood systemic response of breast cancer patients. *Under review* (2017).

[4] Bjørn Fjukstad, Karina Standahl Olsen, Mie Jareid, Eiliv Lund, and Lars Ailo Bongo. 2015. Kvik: three-tier data exploration tools for flexible analysis of genomic data in epidemiological studies. *F1000Research* 4 (2015).

[5] John Gómez, Leyla J García, Gustavo A Salazar, Jose Villaveces, Swanand Gore, Alexander García, Maria J Martín, Guillaume Launay, Rafael Alcántara, Noemi Del Toro Ayllón, and others. 2013. BioJS: an open source JavaScript framework for biological data visualization. *Bioinformatics* (2013), btt100.

[6] R Daniel Kortschak and David L Adelson. 2014. bíogo: a simple high-performance bioinformatics toolkit for the Go language. *bioRxiv* (2014). DOI:https://doi.org/10.1101/005033 arXiv:http://biorxiv.org/content/early/2014/05/12/005033.full.pdf

[7] Keiichiro Ono, Tanja Muetze, Georgi Kolishovski, Paul Shannon, and Barry Demchak. 2015. CyREST: Turbocharging Cytoscape Access for External Tools via a RESTful API. *F1000Research* 4 (2015).

[8] Jeroen Ooms. 2014. The OpenCPU System: Towards a Universal Interface for Scientific Computing through Separation of Concerns. *arXiv preprint arXiv:1406.4806* (2014).

[9] Andrea Sboner, Xinmeng Jasmine Mu, Dov Greenbaum, Raymond K Auerbach, and Mark B Gerstein. 2011. The real cost of sequencing: higher than you think! *Genome biology* 12, 8 (2011), 125.

[10] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. 2003. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research* 13, 11 (2003), 2498–2504.

[11] Christopher L Williams, Jeffrey C Sica, Robert T Killen, and Ulysses GJ Balis. 2016. The growing need for microservices in bioinformatics. *Journal of Pathology Informatics* 7 (2016).

[12] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. *HotCloud* 10, 10-10 (2010), 95.