A peer-reviewed version of this preprint was published in PeerJ on 15 January 2019.

View the peer-reviewed version (peerj.com/articles/6230), which is the preferred citable publication unless you specifically need to cite this preprint.

Almeida JS, Hajagos J, Saltz J, Saltz M. 2019. Serverless OpenHealth at data commons scale—traversing the 20 million patient records of New York's SPARCS dataset in real-time. PeerJ 7:e6230
https://doi.org/10.7717/peerj.6230

Serverless OpenHealth at data commons scale - traversing the 20 million patient records of New York's SPARCS dataset in real-time

Jonas S Almeida [Corresp., 1] , **Janos Hajagos** [1] , **Joel Saltz** [1] , **Mary Saltz** [2]

[1] Biomedical Informatics, State University of New York at Stony Brook, Stony Brook, New York, United States

[2] Radiology, State University of New York at Stony Brook, Stony Brook, New York, United States

Corresponding Author: Jonas S Almeida
Email address: jonas.almeida@stonybrookmedicine.edu

In a previous report, we explored the serverless OpenHealth approach to the Web as a Global Compute space. That approach relies on the modern browser full stack, and, in particular, its configuration for application assembly by code injection. The opportunity, and need, to expand this approach has since increased markedly, reflecting a wider adoption of Open Data policies by Public Health Agencies. Here, we describe how the serverless scaling challenge can be achieved by the isomorphic mapping between the remote data layer API and a local (client-side, in-browser) operator. This solution is validated with an accompanying interactive web application (bit.ly/loadsparcs) capable of real-time traversal of New York's 20 million patient records of the Statewide Planning and Research Cooperative System (SPARCS), and is compared with alternative approaches. The results obtained strengthen the argument that the FAIR reproducibility needed for Population Science applications in the age of P4 Medicine is particularly well served by the Web platform.

Serverless OpenHealth at data commons scale - traversing the 20 million patient records of New York's SPARCS dataset in real-time

Jonas S. Almeida, PhD[1*], Janos Hajagos, PhD[1], Joel Saltz, MD PhD[1], Mary Saltz, MD[2]
[1]Dept Biomedical Informatics; [2]Dept Radiology, Stony Brook University (SUNY), Stony Brook, New York, USA

* jonas.almeida@stonybrookmedicine.edu

Abstract

In a previous report, we explored the serverless OpenHealth approach to the Web as a Global Compute space. That approach relies on the modern browser full stack, and, in particular, its configuration for application assembly by code injection. The opportunity, and need, to expand this approach has since increased markedly, reflecting a wider adoption of Open Data policies by Public Health Agencies. Here, we describe how the serverless scaling challenge can be achieved by the isomorphic mapping between the remote data layer API and a local (client-side, in-browser) operator. This solution is validated with an accompanying interactive web application (bit.ly/loadsparcs) capable of real-time traversal of New York's 20 million patient records of the Statewide Planning and Research Cooperative System (SPARCS), and is compared with alternative approaches. The results obtained strengthen the argument that the FAIR reproducibility needed for Population Science applications in the age of P4 Medicine is particularly well served by the Web platform.

Introduction

Two years ago we approached the feasibility of distributing interactive applications delivered entirely as in-browser constructs [1]. That software ecosystem was then described as "OpenHealth" with reference to the OpenData policy [2]. A multitude of BigData health-related resources have since become available, from the National Institutes of Health such as NCI's Genome Data Commons [3], to Population Health outcomes data collected by the Dept Health of a number of US states such as New York [4]. Specifically, "OpenHealth applications" are assembled by code injection (JavaScript) and hosted with version control as github pages (gh-pages), which decouples the presentation layer from the logistics of data analysis and its governance [1]. That is, there are no servers to be maintained or applications to be downloaded and installed, which greatly extends the lifespan of the computational artifact.

The merits of the "serverless" approach have been well understood, and have been applied to biomedical data for a number of years, from genomics [5] to image analysis in Pathology [6]. However, until recently it came with the suspicion that either the analytical challenge could not computationally intensive, or that a dedicated server-side indexing resource would have to help carry the load. Interestingly, this perception that the performance of the "cloudification" [7] of large data assets is challenged, persists even when confronted with the favorable tabulation of execution times, as with did in that report at AMIA 2016. Instead, this architectural argument appears to be one that requires the development of "believe it when I see it" proof of concept applications that rely exclusively on the API of the data resource along the lines recently detailed for GDC, NCI Genomic Data Commons [3]. This argument, and the development of a validating application, were approached here by targeting Open Health Data resources of the Department of Health of New York state [4]. In that data-intensive infrastructure, the core Data Commons argument that APIs with the ability to consume functionalized query languages are needed is addressed by SoQL [8]. On the one hand, this still falls short of the full Backend-as-a-Service (BaaS) model pursued by Data Commons [9]. On the other, because of the real-world shortcomings of public health data discussed later in this report, the Open Health Data offers the clearest practical assessment of the argument that the BaaS model is viable for any Data resource with a REST API able to consume query languages.

Methods

Architecture

The architecture design for this application starts with OpenHealth [1] , which is about in-browser constructs assembled on-the-fly by code injection, with the primary source of data served by remote HTTP-REST Application Programming Interfaces (API). The original implementation, depicted in ***Figure 1B***, followed the straightforward API Economy model [10] of stateless integration by bringing together data from different sources via REST (Representational State Transfer) APIs. This is also the architecture where the ability to handle large amounts of heterogeneous data comes into question. Recalling from the Background Section, addressing this scaling challenge is best pursued with real-world health data sources, with real-world problems such as the lack of referential integrity that is often encountered in OpenData systems. Those practical challenges, the argument goes, would not be accurately assessed by applications targeting synthetic datasets or targetting heavily engineered BigData such as Data Commons infrastructure.

Data

The data used for this study is that of New York state Statewide Planning and Research Cooperative System (SPARCS) [4], made publicly available by the state's Department of Health via SoQL APIs [8]. As detailed in the program's web page at www.health.ny.gov/statistics/sparcs at the time of this writing, "*SPARCS is a comprehensive all payer data reporting system established in 1979 as a result of cooperation between the healthcare industry and government. The system was initially created to collect information on discharges from hospitals. SPARCS currently collects patient level detail on patient characteristics, diagnoses and treatments, services, and charges for each hospital inpatient stay and outpatient (ambulatory surgery, emergency department, and outpatient services) visit; and each ambulatory surgery and outpatient services visit to a hospital extension clinic and diagnostic and treatment center licensed to provide ambulatory surgery services.*"

The public tier of the SPARCS dataset accessed by accompanying application documents 33 variables covering a range of parameters, from demographic and geographic to clinical, including payment information and identification of caregiver. Figure 2 provides a snapshot of the first entry of the over 2 million records for 2016. As the API section below details, ***this report and the accompanying application do not make any data available***: it simply distributes a in-browser computational artifact that engages the application programming interfaces of the Department of Health on behalf of the user (not the application developer).

API (application programming interface)

Table 1 list all of the SoDA [8] endpoints used by the accompanying application (see Availability). The document in reference details the API specification and the role of Socrata in the configuring the interoperability for Open Data infrastructure. For example, the record displayed in figure 2 can be obtained by dereferencing the address https://health.data.ny.gov/resource/gnzp-ekau.json?$limit=1.

Availability of serverless application

The web application validating the serverless model (Fig.1C) is available at bit.ly/loadsparcs (short link to https://mathbiol.github.io/#load%20sparcs). All code is available with open source and version control, both the base application at https://github.com/mathbiol/mathbiol.github.com and the sparcs module, at https://github.com/mathbiol/sparcs. All dependencies of this software are themselves also open source and, similarly to the accompanying application, only use JavaScript (EcmaScript) to ensure that no downloads or installations are needed. The latter is critical to explore the model where code is able to travel to the computational scope of a user engaging a data source [11]. As discussed below, the unimpeded portability of the application signifies that it explores the scalability of controlled usage. Although the use of the application is what validates the results described in this report, a web cast video demo of traversing the SPARCS data is also available at mathbiol.github.io/sparcs/youtube.

Results

At an architectural level, the SPARCS application was built on the foundations of the OpenHealth serverless model [1]. That architecture corresponds to a cached version of the Web 2.0 AJAX model described in Fig1B. As overviewed in the Background section, the feasibility of that model is typically limited to applications that integrate moderate data volumes by operating the Data layer API in a narrowly prescribed manner. This architecture was changed by creating a client-side object with attributes that map to the query language consumed by SoQL API, as explained in Figure 1C. The key role of the isomorphic mapping of client-side methods to data intensive server-side operations is illustrated in Figure 3 for the count method used to generate the data in Table 1.

The snapshots in figures 3 and 4 illustrate the wide versatility of complex query constraints defined by the operation of the user interface, which is itself assembled in the user's web browser without download or installation. That development versatility is the functionality that enables the BaaS model associated with the architecture described in Figure 1C. However, the full measure of the BaaS model will be the operation of the APIs of remote data intensive resources, as if they were local to the user's own machine. That confirmation of scalability without loss of real-time interaction can only be verified by operating the application. See Availability in the Methods section for the live web-based "serverless" application and demonstrative webcast video. The key role of the asynchronous NoSQL caching in the browser, IndexedDB, for web-based biomedical informatics has been noted by other researchers [12].

Comparison to existing software tools

The development of mobile-first software to traverse open health data is still relatively new. As detailed in our original report on OpenHealth applications [1], this reflects the early stage of development of consumer-facing software for outcomes-driven assessment of Health Care services. The key change is the public availability of large volumes of patient derived data that would have been considered too sensitive for publication just 2 years ago when the original OpenHealth tools were developed. Accordingly, two comparisons to existing tools are in order, speed and interactivity, while engaging the same SoQL API exposed by the Department of Health of the state of New York (health.data.ny.gov). The first comparison is straightforward: dereferencing a standard stateless application such as bit.ly/pqiSuffolk has a much longer assembly time, in the order to tens of seconds to a minute, than the approach presented here (Fig 1C), bit.ly/loadsparcs, which takes less than 10 seconds and traverses a dataset over 100 times larger. The interactivity comparison is not as quantitatively straightforward because it requires the use of the analytical tools published with the data. That exercise can be approached by dereferencing, for example, health.data.ny.gov/Health/All-Payer-Hospital-Inpatient-Discharges-by-Facilit/srur-4jdu, and noting that the numerical results are not themselves linked to additional analysis where they are used as independent variables. In summary, the proposed engagement of the data intensive patient derived SPARCS dataset has a clear advantage over approaches that do not use the cached BaaS model. That advantage is proposed here as a definite argument to approach data intensive software Commons for research applications by using this model. That is, by mapping server-side to client-side abstractions as a generic backend that goes beyond the conventional stateless architecture of REST APIs. That conclusion, discussed at length in the next section, is particularly well aligned with recent developments in funding agencies promoting the use of interoperable cloud-hosted Research Commons infrastructure.

Discussion

The objective of this coding exercise was to assess the viability of real-time traversal of real-world large health data resources. Lack of referential integrity caused by loose controlled vocabularies is amongst the most common and most challenging. Solving this problem *ex post* [13] in the presentation layer (in this case in the browser) is often considered an opeless exercise because of the large number of records that would have to be fixed on-the-fly. Instead, mending referential integrity is typically addressed with ETL processes running in the data center. However, that objection may no longer be as relevant, because JavaScript engines have improved to the point of measuring themselves favourably with compilers in more conventional Data Science platforms. Case in point, close inspection

of the SPARCS module reveals the use of MapReduce functional patterns, which may be executed in the machine's Graphic Processing Units (GPU). It is noteworthy that modern browser includes native GPU APIs as part of its Document Object Model (DOM). It should also be noted that referential integrity in the SPARCS dataset is, as feared, broken by both loose variable naming conventions and value binning. To fix it, extensive corrections via Map operations are embedded in the *sparcs.getJSON* read operator, as detailed in the source code at https://github.com/mathbiol/sparcs/blob/master/sparcs.js#L34. In spite of the on-the-fly computation, there is no noticeable loss of interactivity of the SPARCS user-interface. Although not attempted here, this programmatic approach could be replaced by a more formal, declarative, approach to "sloppy data integration" [14].

The Backend-as-a-Service (BaaS) model advanced by recent Data Commons infrastructure [9] are recognized as the scalable route towards Precision Medicine [15]. Therefore, what combination of API language and query engine would best serve that goal in a FAIR manner [16] is a critical design goal. In this study, SoQL (see Methods) was found to provide the necessary read-only interoperability. Naturally, the full BaaS model would require a more comprehensive approach to schema definition and data presentation. While this discussion is beyond the scope of the present report, it may be informative to note that data submission to NCI Genomic Data commons, at the time of this writing (as per GDC v1.13.0, Feb 18, 2018), requires the use of GraphQL as the interoperability model of choice for 3rd generation Data Commons infrastructure [17].

Conclusion

The use of in-browser "serverless" applications (Web Apps calling data layer APIs directly) was tested with the real-world challenge of assembling web applications capable of traversing 20 million patient records of the public SPARCS dataset served by New York's Department of Health. The portability and security of the web app model is a good match to the principles of FAIR Data Commons. The real-world test was that of interactive and open-ended constraint satisfaction on this large Data Space of well over half a billion individual measurements (33x19,907,183 = 656,937,039), convoluted by a significant lack of referential integrity. In spite of these obstacles, the isomorphic mapping of client-side operators to remote APIs supporting a full fledged query language, combined with the native support for vectorized operators of the modern Web browser, was shown to achieve the performance levels required for real-time interactivity. It is therefore concluded that the emerging Data Commons frameworks are particularly well suited for ecosystems of Web Applications. This BaaS behaviour suggests a solution that overcomes the need for local, or even on-premise, implementations of Biomedical Informatics applications.

References

1. Almeida, J. S. *et al.* OpenHealth Platform for Interactive Contextualization of Population Health Open Data. *AMIA Annu. Symp. Proc.* **2015,** 297–305 (2015).
2. Burwell, S. M., VanRoekel, S., Park, T. & Mancini, D. J. Memorandum for the Heads of Executive Departments and Agencies - Managing Information as an Asset. (2013). Available at: https://project-open-data.cio.gov/policy-memo. (Accessed: 6th March 2018)
3. Wilson, S. *et al.* Developing Cancer Informatics Applications and Tools Using the NCI Genomic Data Commons API. *Cancer Res.* **77,** e15–e18 (2017).
4. NY. State of New York | Open Data Health | Health Data NY. *New York State Department of Health | Health Data NY* (2018). Available at: https://health.data.ny.gov/. (Accessed: 5th March 2018)
5. Wilkinson, S. R. & Almeida, J. S. QMachine: commodity supercomputing in web browsers. *BMC Bioinformatics* **15,** 176 (2014).
6. Almeida, J. S. *et al.* ImageJS: Personalized, participated, pervasive, and reproducible image bioinformatics in the web browser. *J. Pathol. Inform.* **3,** 25 (2012).
7. Bremer, E., Kurc, T., Gao, Y., Saltz, J. & Almeida, J. S. Safe 'cloudification' of large images through picker APIs. *AMIA Annu. Symp. Proc.* **2016,** 342–351 (2016).

8. Socrata. API Endpoints to Socrata Open Data Infrastructure. (2018). Available at: https://dev.socrata.com/docs/endpoints.html. (Accessed: 8th March 2018)
9. Grossman, R. L., Heath, A., Murphy, M., Patterson, M. & Wells, W. A Case for Data Commons: Toward Data Science as a Service. *Comput. Sci. Eng.* **18,** 10–20 (2016).
10. Brown, A., Fishenden, J. & Thompson, M. *Digitizing Government: Understanding and Implementing New Digital Business Models*. (Palgrave Macmillan UK, 2014).
11. Bell, G., Hey, T. & Szalay, A. Computer science. Beyond the data deluge. *Science* **323,** 1297–1298 (2009).
12. Shi, X. *et al.* PopGeV: a web-based large-scale population genome browser. *Bioinformatics* **31,** 3048–3050 (2015).
13. Hoekstra, R. The knowledge reengineering bottleneck. *Semantic Web* **1,** 111–115 (2010).
14. Almeida, J. S. *et al.* Data integration gets 'Sloppy'. *Nat. Biotechnol.* **24,** 1070–1071 (2006).
15. Jensen, M. A., Ferretti, V., Grossman, R. L. & Staudt, L. M. The NCI Genomic Data Commons as an engine for precision medicine. *Blood* **130,** 453–459 (2017).
16. Wilkinson, M. D. *et al.* The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data* **3,** 160018 (2016).
17. Grossman, R. Gen3 Software. *Center for Data Intensive Science* (2018). Available at: https://cdis.uchicago.edu/gen3/. (Accessed: 8th March 2018)

Figure 1 (on next page)

Figure 1 - Evolving Web Computing Architectures

Evolution of the API economy from its pre-REST stage (A) to stateless transfer via HTTP (B), recently abstracted by constructs like GraphQL that combine an API language with a query engine (C). The prototype accompanying this report uses SoQL (see Methods) to illustrate the viability of the latter design, where the traversal of the Data Layer is abstracted as a stateless backend. The Cloud instantiation of this model approaches the description of BaaS (Backend-as-a-service).

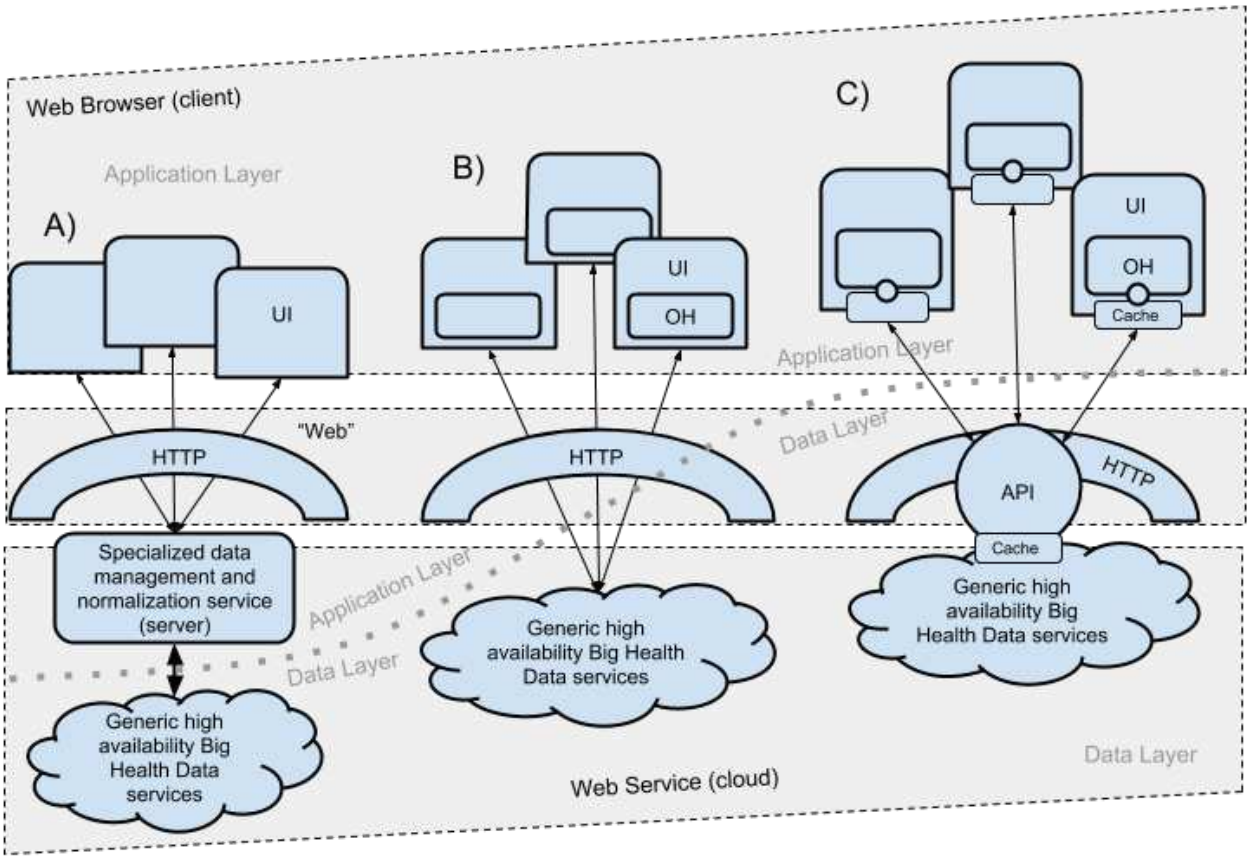

Web Browser (client)
Application Layer
A)
UI
B)
UI
OH
C)
UI
OH
Cache
Application Layer
Data Layer
"Web"
HTTP
HTTP
API
HTTP
Cache
Specialized data management and normalization service (server)
Application Layer
Data Layer
Generic high availability Big Health Data services
Generic high availability Big Health Data services
Generic high availability Big Health Data services
Data Layer
Web Service (cloud)

Figure 2 (on next page)

Figure 2 - Snapshot of first of the 2,343,429 public records for 2016.

See Table 1 for the full count. See also API section below for more information about why this exact public record can be programmatically retrieved from NY state Dept of Health: https://health.data.ny.gov/resource/gnzp-ekau.json?$limit=1

.

```
{
  "apr_drg_description" : "Other pneumonia",
  "gender" : "F",
  "ethnicity" : "Not Span/Hispanic",
  "zip_code_3_digits" : "147",
  "abortion_edit_indicator" : "N",
  "apr_mdc_description" : "Diseases and Disorders of the Respiratory System",
  "ccs_procedure_code" : "0",
  "apr_severity_of_illness_description" : "Moderate",
  "payment_typology_1" : "Medicare",
  "discharge_year" : "2016",
  "apr_medical_surgical_description" : "Medical",
  "payment_typology_2" : "Private Health Insurance",
  "total_charges" : "3913.23",
  "ccs_diagnosis_description" : "Pneumonia (except that caused by tuberculosis or sexually transmitted disease)",
  "type_of_admission" : "Urgent",
  "facility_id" : "37",
  "ccs_diagnosis_code" : "122",
  "apr_mdc_code" : "4",
  "race" : "White",
  "health_service_area" : "Western NY",
  "age_group" : "70 or Older",
  "facility_name" : "Cuba Memorial Hospital Inc",
  "apr_severity_of_illness_code" : "2",
  "apr_drg_code" : "139",
  "patient_disposition" : "Home or Self Care",
  "ccs_procedure_description" : "NO PROC",
  "attending_provider_license_number" : "90335341",
  "birth_weight" : "0",
  "hospital_county" : "Allegany",
  "total_costs" : "3466.83",
  "length_of_stay" : "3",
  "operating_certificate_number" : "0226700",
  "apr_risk_of_mortality" : "Moderate",
  "emergency_department_indicator" : "N"
}
```

Table 1 (on next page)

Table 1 - Year, record count and public SPARCS data source traversed by the accompanying application.

As the use of the application will make clear, these records come from all 58 counties of the state of New York.

Table 1 - Year, record count and public SPARCS data source traversed by the accompanying application. As the use of the application will make clear, these records come from all 58 counties of the state of New York.

Year	# records	URL
2009	2,665,414	https://health.data.ny.gov/resource/s8d9-z734
2010	2,622,133	https://health.data.ny.gov/resource/dpew-wqcg
2011	2,589,121	https://health.data.ny.gov/resource/n5y9-zanf
2012	2,544,543	https://health.data.ny.gov/resource/rv8x-4fm3
2013	2,428,500	https://health.data.ny.gov/resource/tdf6-7fpk
2014	2,367,283	https://health.data.ny.gov/resource/pzzw-8zdv
2015	2,346,760	https://health.data.ny.gov/resource/82xm-y6g8
2016	2,343,429	https://health.data.ny.gov/resource/gnzp-ekau
total:	19,907,183	https://www.health.ny.gov/statistics/sparcs

Figure 3(on next page)

Figure 3 - Snapshot of the SPARCS module loaded in Google Chrome Web browser with the developer tools open.

These tools are used here to the inspect a client-side method operating a SoQL query across all 8 API endpoints (2009-2016) at NY's Dept of Health. The upper left corner shows the execution in the MathBiol console; The middle left-hand shows the same operator used to generate a list in HTML; The upper right-hand corner shows the code behind the count command, which migrated to the user's browser from mathbiol.github.io/sparcs/sparcs.js (see Availability in Methods); Finally, the middle right-hand corner shows the same command being recognized after negotiating variations in the syntax ("TypeError") used to call it. For clarity, the programmatic count call resulting from "assuming sparcs.count()" is executed manually at the end of that negotiation.

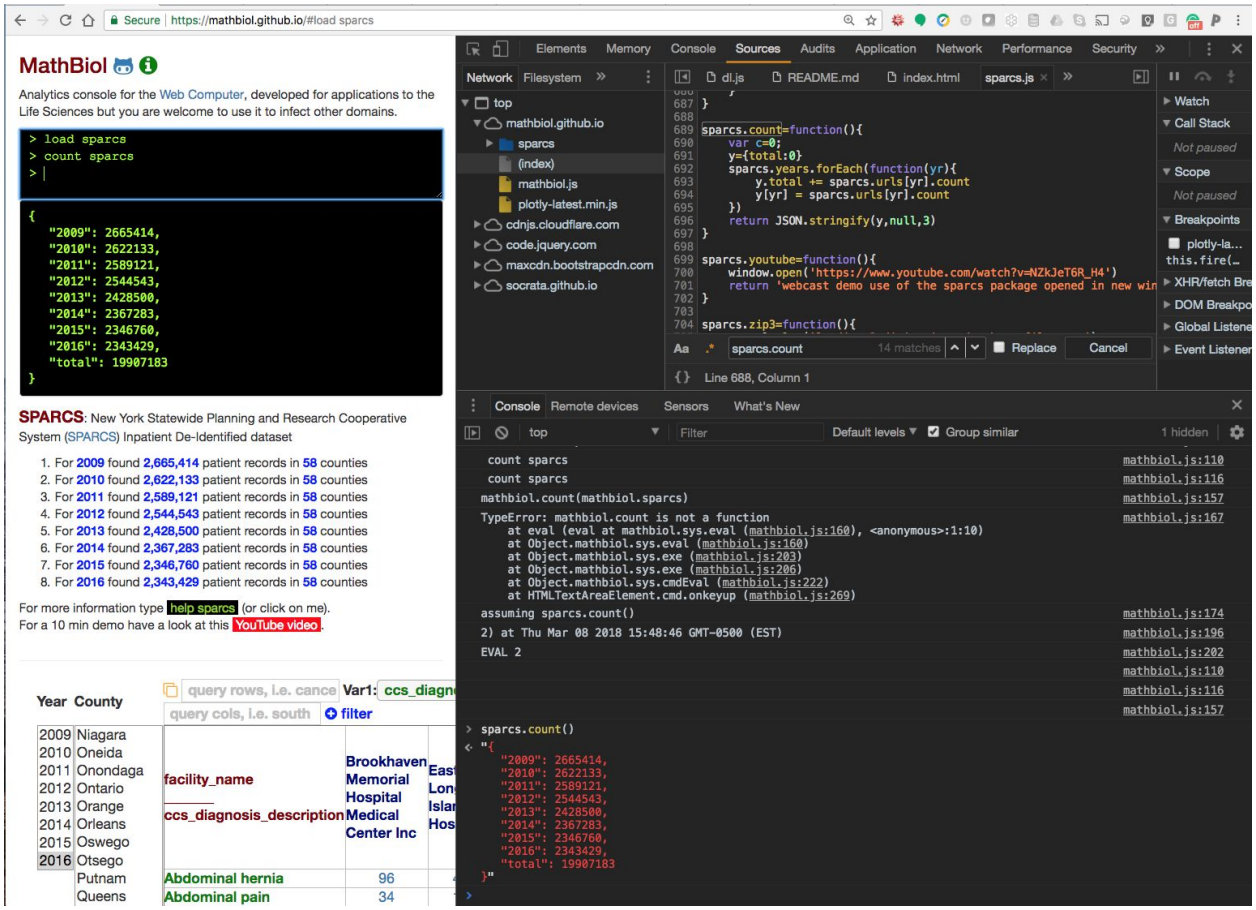
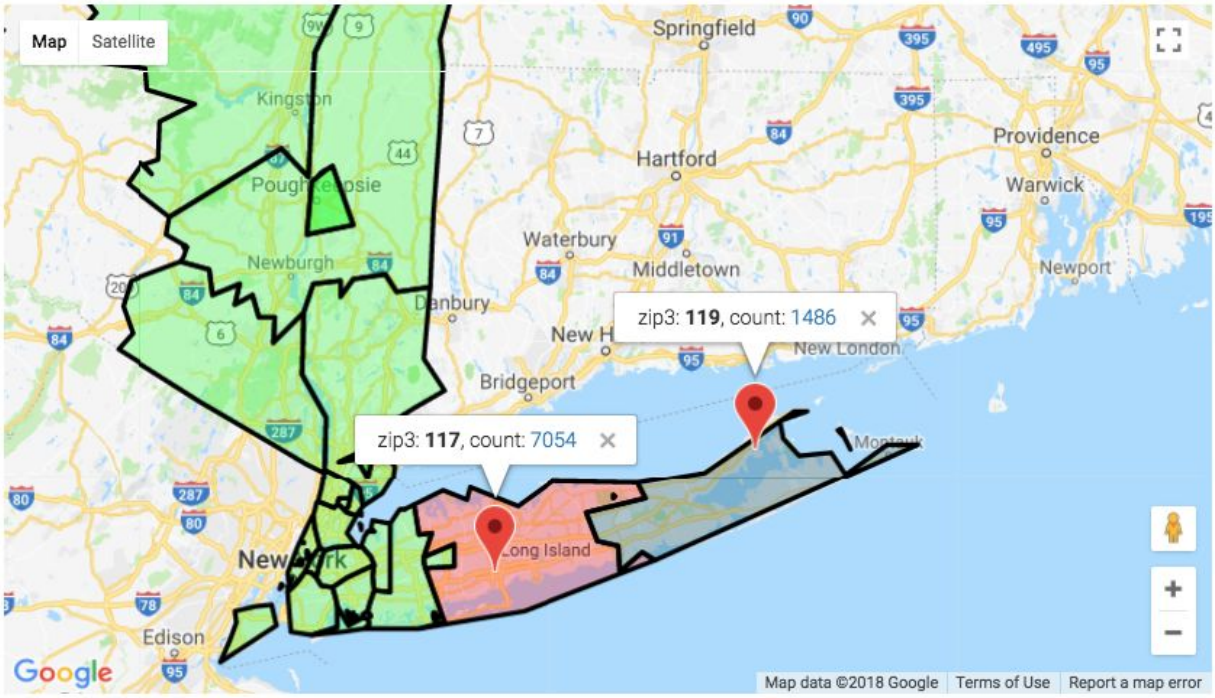

Secure | https://mathbiol.github.io/#load sparcs
MathBiol
Analytics console for the Web Computer, developed for applications to the Life Sciences but you are welcome to use it to infect other domains.
> load sparcs
> count sparcs
{
"2009": 2665414,
"2010": 2622133,
"2011": 2589121,
"2012": 2544543,
"2013": 2428500,
"2014": 2367283,
"2015": 2346760,
"2016": 2343429,
"total": 19907183
}
SPARCS: New York Statewide Planning and Research Cooperative System (SPARCS) Inpatient De-Identified dataset
1. For 2009 found 2,665,414 patient records in 58 counties
2. For 2010 found 2,622,133 patient records in 58 counties
3. For 2011 found 2,589,121 patient records in 58 counties
4. For 2012 found 2,544,543 patient records in 58 counties
5. For 2013 found 2,428,500 patient records in 58 counties
6. For 2014 found 2,367,283 patient records in 58 counties
7. For 2015 found 2,346,760 patient records in 58 counties
8. For 2016 found 2,343,429 patient records in 58 counties
For more information type help sparcs (or click on me).
For a 10 min demo have a look at this YouTube video.
Year County
query rows, i.e. cance
Var1: ccs_diagn
query cols, i.e. south
filter
2009 Niagara
2010 Oneida
2011 Onondaga
2012 Ontario
2013 Orange
2014 Orleans
2015 Oswego
2016 Otsego
Putnam
Queens
facility_name
ccs_diagnosis_description
Brookhaven Memorial Hospital Medical Center Inc
Abdominal hernia
96
Abdominal pain
34
Elements Memory Console Sources Audits Application Network Performance Security
Network Filesystem
top
mathbiol.github.io
sparcs
(index)
mathbiol.js
plotly-latest.min.js
cdnjs.cloudflare.com
code.jquery.com
maxcdn.bootstrapcdn.com
socrata.github.io
dl.js README.md index.html sparcs.js
sparcs.count=function(){
var c=0;
y={total:0}
sparcs.years.forEach(function(yr){
y.total += sparcs.urls[yr].count
y[yr] = sparcs.urls[yr].count
})
return JSON.stringify(y,null,3)
}
sparcs.youtube=function(){
window.open('https://www.youtube.com/watch?v=NZkJeT6R_H4')
return 'webcast demo use of the sparcs package opened in new win
}
sparcs.zip3=function(){
Aa .* sparcs.count 14 matches Replace Cancel
{} Line 688, Column 1
Watch
Call Stack
Not paused
Scope
Not paused
Breakpoints
plotly-la...
this.fire(...
XHR/fetch Bre
DOM Breakpo
Global Listene
Event Listener
Console Remote devices Sensors What's New
top
Filter
Default levels
Group similar
1 hidden
count sparcs mathbiol.js:110
count sparcs mathbiol.js:116
mathbiol.count(mathbiol.sparcs) mathbiol.js:157
TypeError: mathbiol.count is not a function mathbiol.js:167
at eval (eval at mathbiol.sys.eval (mathbiol.js:160), <anonymous>:1:10)
at Object.mathbiol.sys.eval (mathbiol.js:160)
at Object.mathbiol.sys.exe (mathbiol.js:203)
at Object.mathbiol.sys.exe (mathbiol.js:206)
at Object.mathbiol.sys.cmdEval (mathbiol.js:222)
at HTMLTextAreaElement.cmd.onkeyup (mathbiol.js:269)
assuming sparcs.count() mathbiol.js:174
2) at Thu Mar 08 2018 15:48:46 GMT-0500 (EST) mathbiol.js:196
EVAL 2 mathbiol.js:202
mathbiol.js:110
mathbiol.js:116
mathbiol.js:157
> sparcs.count()
< "{
"2009": 2665414,
"2010": 2622133,
"2011": 2589121,
"2012": 2544543,
"2013": 2428500,
"2014": 2367283,
"2015": 2346760,
"2016": 2343429,
"total": 19907183
}"
>

Figure 4(on next page)

Figure 4 - Snapshot of the SPARCS module in portrait mode in a mobile device, illustrating the ability to quickly resolve complex queries in moderately powered devices.

Note how the graphic type responds to the data type: for example, the 3-digit zip code is matched by a geographic map display instead of a bar graph as in Figure 3. The choice of variables can be compounded with additional constraints (Additional filter), in order to, in this example, obtain the age groups and place of residence for patients seen at Stony Brook University Hospital. Each of the count numbers, underlined in blue, is a live link to the corresponding patient cohort.For example, clicking on “ 7054 ” either on the table or in the map will automatically retrieve the full data subset, with the values of all 33 parameters (Figure 2) for each the 7,054 patients.

```
plot zip_code_3_digits for 70 or Older
```



query rows, i.e. cance **Var1:** zip_code_3_digits

Var2: age_group query cols, i.e. south **remove filter**

Additional filter:

facility_name="University Hospital"

Year **County**

2009 Niagara
2010 Oneida
2011 Onondaga
2012 Ontario
2013 Orange
2014 Orleans
2015 Oswego
2016 Otsego
Putnam
Queens
Rensselaer
Richmond
Rockland
Saratoga
Schenectady
Schoharie
Schuyler
Steuben
St Lawrence
Suffolk

age_group / zip_code_3_digits	0 to 17	18 to 29	30 to 49	50 to 69	70 or Older
100	19	9	11	18	20
101	3	1	1	0	3
103	1	1	1	7	3
104	8	4	1	10	6
105	4	4	1	4	3
106	2	0	1	0	0
107	2	1	3	0	2
108	0	1	0	0	0
109	0	5	1	1	1
110	7	2	7	8	9
111	0	0	1	1	7
112	7	14	5	17	22
113	8	7	8	24	19
114	5	5	9	15	15
115	30	28	26	46	53
116	4	1	3	3	3
117	5239	2472	4564	6258	7054
118	7	6	10	9	9
119	1929	880	1356	1733	1486
120	0	0	2	0	0