VCF2JSON: A data migration tool from VCF to JSON

## Abstract

The development of the next-generation sequencing (NGS) technologies has led to massive amounts of VCF (Variant Call Format) files, which have been the standard formats developed with [1000 Genomes Project](https://en.wikipedia.org/wiki/1000_Genomes_Project). At the same time, as a completely language-independent and cross-platform text format, JSON (JavaScript Object Notation) format has attracted considerable interests in the bioinformatics community. Currently, many practical database applications, such as dbSNP, etc., are transitioning to the new design by using JSON to support efficient massive data expansion. In this scenario, a user-friendly mapping tool that supports the data migrations from VCF files to JSON files is greatly needed. Therefore, we present VCF2JSON, a standalone, cross-platform and freely available desktop software for mapping VCF to JSON in this paper. VCF2JSON is developed with Python and supports multi-process. By providing a user-friendly graphical user interface (GUI), users without programming skills could directly deploy this mapping tool in their personal computers or servers and perform the data migration operation by directly running the executable program. VCF2JSON is freely available for downloading from <https://github.com/lyotvincent/vcf2json>.

## Introduction

The next-generation sequencing (NGS) has been widely used, and it dramatically reduces sequencing costs and increases efficiency by parallelizing the sequencing process. The NGS technology is a revolution to traditional sequencing for its low-cost and high-efficiency. The advent of NGS platform led to massive amounts of data. In order to effectively represent these data, a variety of file formats appears. As one of the standard formats developed with [1000 Genomes Project](https://en.wikipedia.org/wiki/1000_Genomes_Project) [1], VCF (Variant Call Format) is widely adopted both in academia and industry.

A growing number of VCF-based applications, frameworks and workflows[2-6] designed for processing next-generation DNA sequencing data have become available. Based on the NGS data, several analysis tools [7-9] are proposed. For example, VCFtools[7] provides a toolset for parsing, analyzing and manipulating VCF files. GQT[8] uses command lines for indexing and querying large-scale genotype data sets in VCF files. Pyvcf and cyvcf2[9] are Python libraries and software packages for parsing and querying of VCF data. These tools usually require experienced users with sufficient programming skills to process VCF data, because of the lack of user-friendly graphical interface to directly manipulate the text-based[10] VCF file, which is an obstacle for the users without programming skills. Recently, more and more platforms (e.g., BrowseVCF[11], VCF-Miner[12], Varsifter[13]) are developed with a user-friendly GUI (Graphical User Interface), which enable ordinary users to search and filter millions of variants in a few seconds.

At the same time, as a cross-platform text format, JSON (JavaScript Object Notation) format has attracted considerable interests in the bioinformatics community. JSON is a lightweight data-interchange format which is easy for humans to read and write[14], and is also easy for machines to parse and generate, which has led to many JSON-based tools with high performance when process large-scale data. Furthermore, JSON is a completely language-independent[15] text format and there are many tools developed base on the JSON format including NoSQL database MongoDB[16] which is an open-source cross-platform and support distributed processing of large data sets[17]. These properties make the JSON format widely used in several operating systems and the most common formats for data interchange[18] in Web. JSON is an ideal data model for massive data management[19]. Therefore, many practical database applications, such as dbSNP, are transitioning to the new design by using JSON to support future massive human variation data expansion. The emergence of new design of file format often involves data migrations from the original data file to the new one. There are several works about the data migrations such as VCF2RDF[20], BGT[21], wormtable[22], etc. For example, the move of computational genomics workflows to Cloud Computing platforms leads to a semantic mapping tool named VCF2RDF[20], which is a web application that provide an isomorphic map between VCF and the RDF (Resource Description Framework). And now, the new design of JSON-based applications naturally leads to the need for the data migration from the original data file to the JSON.

In fact, the explosion growth of data in recent years leads to more applications that choose JSON format files to store massive data sources, since JSON files could be easily imported into MongoDB to support high scalability[23]. In addition, it is often difficulty to directly read VCF files with a large size of data, while JSON file has better readability and it is much easier to parse[24][25] and understand. Considering these advantages of using JSON, in this paper, we present a data migration tool (called VCF2JSON) used to achieve the data migrations from VCF to JSON to effectively support future massive data expansion, annotation and management. In particular, VCF2JSON is a standalone, cross-platform and freely available desktop software, which could be easily used by ordinary users without programming skills. To the best of our knowledge, this is the first publicly available GUI tool that deals with the data migrations from VCF files to JSON files. The core of VCF2JSON is developed with Python, and its GUI framework is developed with Electron. VCF2JSON could be deployed at ordinary computers or servers. Owing to reading VCF files by chunks, our tool does not need to take the whole VCF file inside the memory and could process the large size of files without running out of memory. We verify the effectiveness of the proposed tool by using the chromosomes from the 1000 Genome Project ([www.internationalgenome.org](http://www.internationalgenome.org)) in Phase 3, total 25 gzip-compatible compressed files, including 22 autosomes, XY chromosomes and MT chromosome, and shows the high efficiency in processing large-scale VCF data with multi-process.

## MATERIALS AND METHODS

### The VCF and the mapping rule

The VCF has become the standard format to store variations. A VCF file could be divided into three sections[26], which are called meta-information lines, a header line and data lines[27]. Each meta-information line starts with “##” and must be “key=value” pairs. The header line starts with “#” and names of the 8 fixed, mandatory columns. If genotype data is given, they are followed by a FORMAT column header, then an arbitrary number of sample IDs. The header line is tab-delimited. In the data section, there are 8 fixed fields per record and data lines are tab-delimited. Missing values are specified with a dot (‘.’). The first eight columns describe the chromosome(CHROM), position(POS), variant(ID) that is a key resource identifier in dbSNP[28], reference base of the variation(REF), alternate base of the variation (ALT), variation quality(QUAL), filter status(FILTER), variant annotation(INFO), and the last column is genotype fields which contain all samples.

图片去掉箭头 加上AB genotype换成 samples

In the following, a set of mapping rules are given to deal with the data migration from VCF to JSON.

***Rule 1.*** *For meta-information lines and the header line, a root element named Header is created in the corresponding JSON. Similarly, for data lines, the root element name is Data in JSON as shown in Figure 1.*

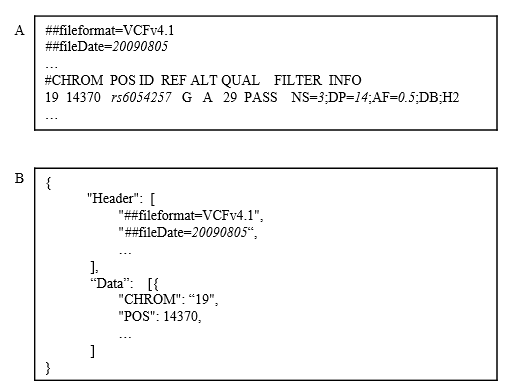


Figure 1: a mapping example of Rule 1, (A) VCF and (B) JSON

***Rule 2.*** *For each missing value that is specified with a dot (‘.’) in VCF, let it be set to empty (‘’) when the value is string, while let it be ‘-1’ when the value is number.*

***Rule 3.*** *When the field (F) contains multiple keys, we can create the elements of these keys to be under the element for F as child elements, which form a nested structure. When the field (F) contains multiple values, we can turn F into the array.*

***Rule 4.*** *For each element in FILTER, give it a “Filter\_” as prefix. After that, each element and a Boolean variable form a key-value pair and then placed as a child of FILTER. If the filter status is passed, the “Filter\_pass=true” pair will be used, while, if not, the “Filter\_pass=false” pair will be used.*

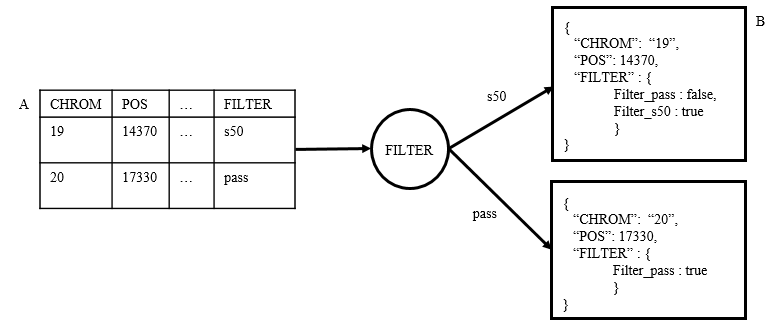


Figure 2: an example of Rule 4, (A) VCF and (B) JSON

***Rule 5.*** *For the genotype field, a root element named SAMPLES is created in the corresponding JSON. And the SAMPLES maps an array where each sample is stored in. In this array, a key element named SampleNo is defined as sample ID as shown in Figure 3.*

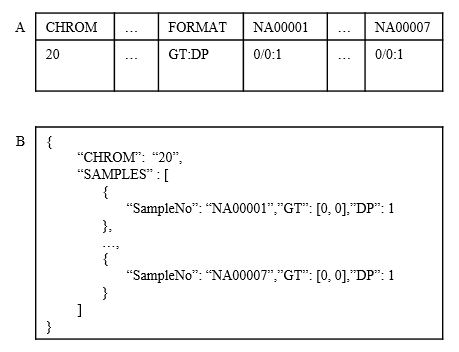


Figure 3: an example of Rule 5, (A) VCF and (B) JSON

Algorithm 1 shows the transformation algorithm based on the mapping rules above. In particular, VCF2JSON exacts every line from the VCF file and then transforms it into JSON. Firstly, the parameters are initialized, where the line number (*i*) is initialized with 0, the iterator (*D*) is treated as chunks from the VCF file and the string (*Header*)is initialized using *Rule 1*. In the main loop (lines 5-24), program processes one row of VCF data each time until the last line is processed. In the first loop (lines 6-8), the first six columns of VCF is extracted, transformed and saved in [temporary variable](http://www.baidu.com/link?url=pW8muJa8UrN_tYpcOA7lNI0shMf7kEs--avSbtTP-u4Mc_Ee_vG-aXgPtm1_ezTjC_s5vd_sHaJrYvLnou5aW61YJ_kxX7pPWmoeuPVPs4RIfTDDPNLMZAUYBajKyeht) (*JsonData*) using *Rule 2,3*. Similarly, in the following loops (lines 10-20) by applying *Rule 2,3,4,5*, the FILTER, INFO and GENOTYPE are also transformed and appended in variable (*JsonData*). Finally, we will get a JSON string that stored one line of the variant data, and then write JSON string to the file. As soon as the last line of the input VCF file is transformed, we will obtain the mapped JSON file. The algorithm runs in O(*ki*) time where *i* is the number of data lines and *k* is the number of data columns in the VCF file.

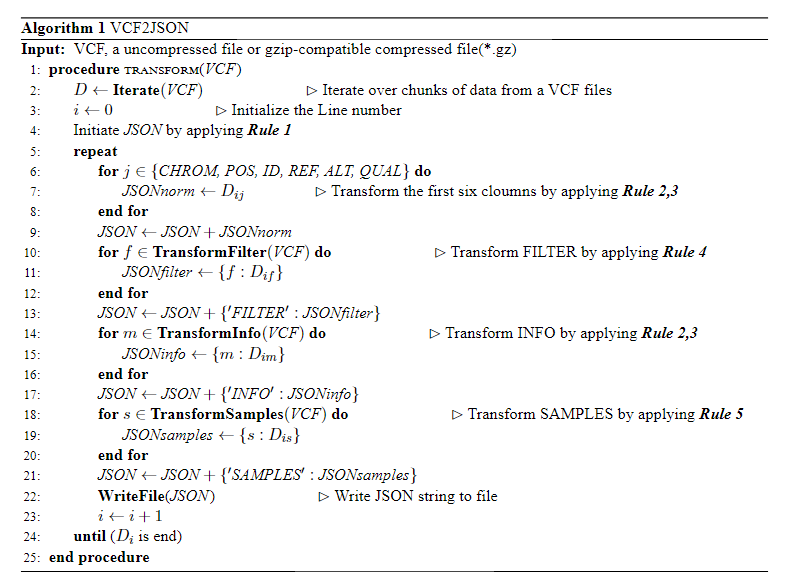


Figure 4 shows a transformation example, where the first section named Header in JSON file saves the header information of the original VCF file. The second section named Data is an array, which contains the corresponding transformed data. Each element in the JSON array represents one variant in VCF.

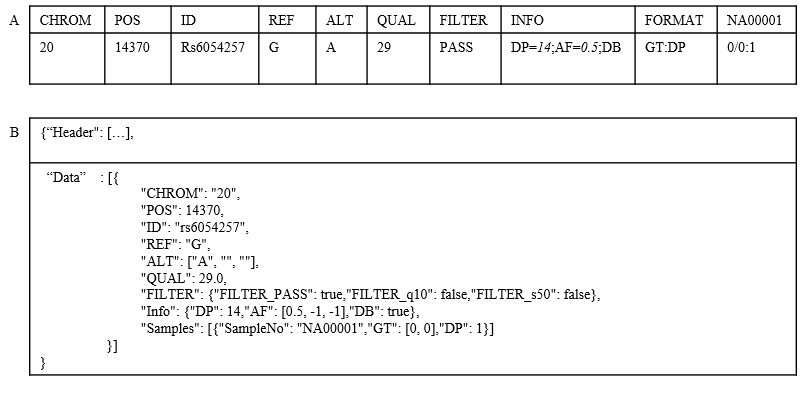


Figure 4: a data migration sample, (A) VCF and (B) JSON

### Loading and processing VCF data

In the loading step, with a user-friendly GUI, a user selects a VCF file from a pop-up dialog box, and then VCF2JSON shows the VCF file path at an input label. After that, user can choose an output directory or use the default output directory. After clicking the Go button, data migrations from the VCF file to the JSON file are processed. VCF2JSON accepts both uncompressed and gzip-compatible compressed (\*.gz) VCF files as inputs. We also provide the command line tool for processing multi-files, which is provided in transform\_command.py. Users could user their personal computers with a python 3.6 environment and install the package provided by VCF2JSON to accomplish the data migration.

Because each line of data in the VCF file is independent, VCF2JSON reads VCF by chunks, and multiple processes could be used when processing a VCF file, which is benefit to provide a high performance when processing big VCF files. To make sure the process performed normally and reduce process switching, the number of processes is set to *p* (for *p* = *CPUs*/2, *CPUs* means the number of CPUs). Each time *r* (*r=5k\*p*, *p* means the number of processes) rows of data are read and assign to all child processes for processing. Consequently, after loading *r* rows of data and writing them to the file, the memory will be released, in this scenario, VCF2JSON will not run out of memory when process very large files.

### Software Framework and Implementation

The basic framework of VCF2JSON is shown in Figure 5. The core of VCF2JSON is built with Python (v3.6) program. We use scikit-allel (https://github.com/cggh/scikit-allel), a Python package for parsing VCF. The framework of the GUI is designed and implemented with Electron (<https://electronjs.org/>) which using JavaScript, HTML and CSS to write cross-platform desktop application. The user interface page architecture, design and functionality, is developed using several JavaScript libraries as follows. The main window page is produced by jQuery 3.3.1 (<https://jquery.com/>), Bootstrap 4.1.3 (<https://getbootstrap.com/>) and [font Awesome 4.7.0](file:///C:\Users\quz\AppData\Roaming\Microsoft\Word\font%20Awesome%204.7.0) (<https://fontawesome.com/>). The preview page is produced by DataTables 1.10.18 (<https://datatables.net/>). The frontend and backend communicate with each other is supported by Zerorpc (<http://www.zerorpc.io/>) that uses PC port for communication. The software has been tested in several operating systems (Unix, Mac and Windows).

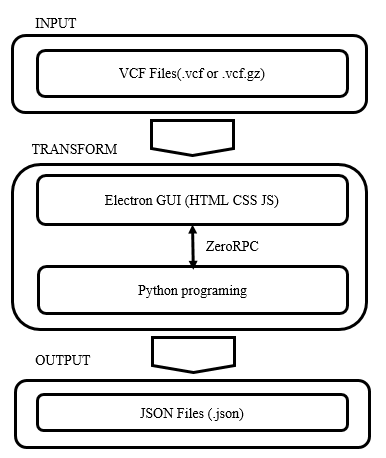
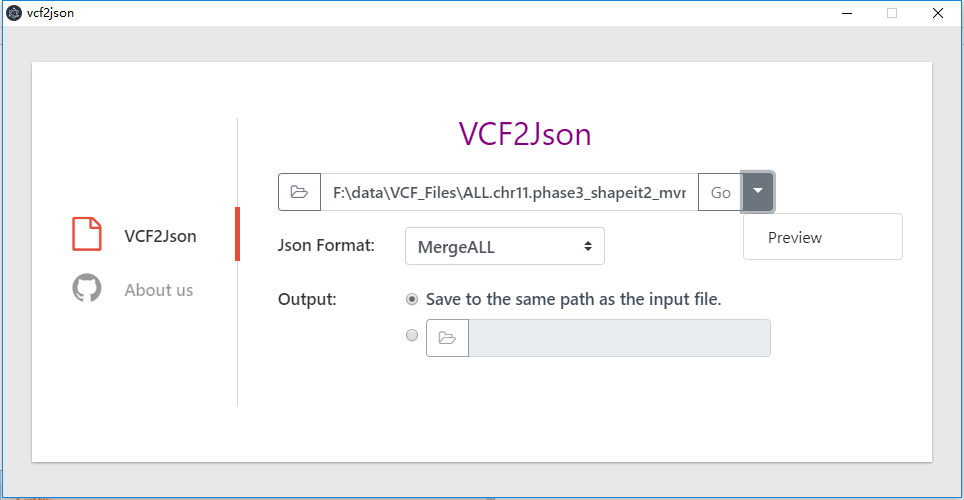
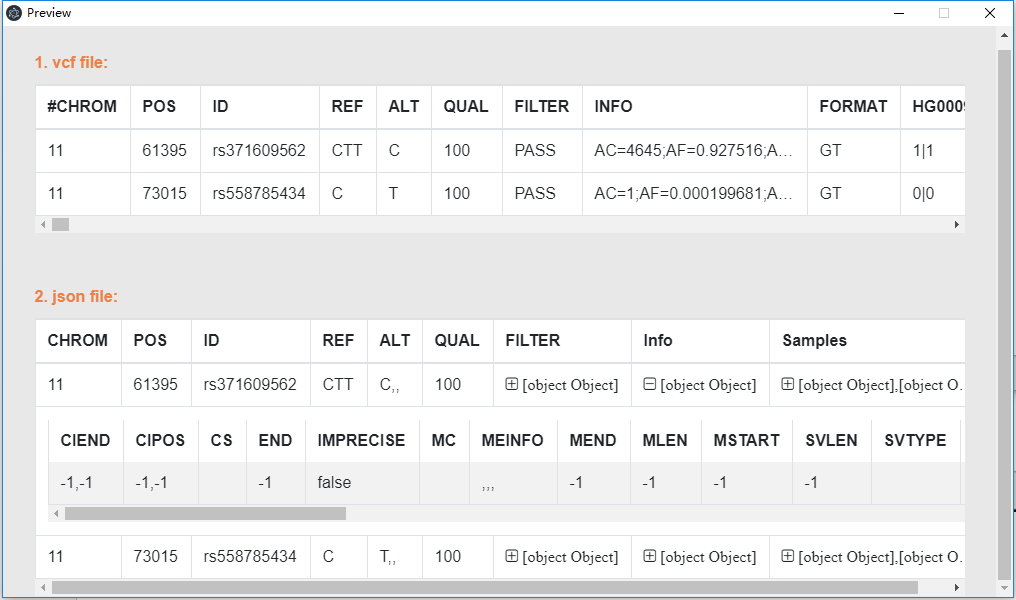


Figure 6: software framework

### Application of VCF2JSON: a utility case

The data migration is initiated from the main page (Figure 6A). VCF2JSON accept uncompressed or gzip-compatible compressed (\*.gz) VCF files, users can open pop-up file dialog and select a VCF file or compressed VCF file, and then the VCF file path will be shown. In ‘JSON Format’ label, there are two options (‘MergeAll’ and ‘MergeSamples’) for users to select. One is ‘MergeAll’, when users choose this option, VCF2JSON will use nested structures to build the JSON file. Some fields may contain a lot of data, after the data migration, data obtained from the same field will be integrated, e.g., FILTER, INFO and GENOTYPE. In the JSON file, these three fields represent three keys, for instance, the section will be “{‘FILTER’ :{…}, ‘INFO’ :{…}, ‘SAMPLES’ :{…}}”. The other is ‘MergeSamples’, where the nested structures are used in FILTER and SAMPLES, that is to say, data obtained from INFO will not be integrated after data migrations. For example, the corresponding section in JSON will be “{‘FILTER’ :{…}, ‘AC’: 1,…,‘AF’: 1, ‘SAMPLES’ :{…}}”. Currently, two JSON formats are available for the user and we could also provide a custom JSON format according to the users’ feedback in the future. When users click the “GO” button and they will obtain a transformed JSON file at the output directory. The execution times of data migrations depend on the input file size, file structure and the used numbers of CPUs. In addition, VCF2JSON provides a preview function. After selecting a VCF file, users could preview parts of the transformed files by clicking the preview button (Figure 6B).



Select JSON format

Preview first two rows

A

B

Figure 7: Main operation and preview interfaces, (A) Main interface for VCF2JSON and (B) Preview of the transformations

## Results

### Performance of VCF2JSON

To evaluate the performance of VCF2JSON, we use the human chromosomes obtained from the 1000 Genome Project ([www.internationalgenome.org](http://www.internationalgenome.org)) in Phase 3, which contains 25 gzip-compatible compressed files, including 22 autosomes, XY chromosomes and MT chromosome. Our experiment is tested on the server running Linux 4.4.0-130-generic (64 bit) with 64 CPUs (Intel Xeon CPU E7- 4820 @ 2.00GHz) and 1TB memory. All operations were performed using 20 CPUs. Note that the reason why not use half the number of CPUs to run data is that others are using the server at the same time, in order to make sure that the CPU performed normally, only 20 CPUs are used.

The experimental results of the 25 files above are shown in Table 1, where the number of samples in VCF is 2535 and the number of annotations is 4 in per variant on chromosomes except “ChrX”. From Table 1, we can see that the execution times depend on the numbers of samples, variants, etc. Obviously, the execution times increase when the numbers of the input samples and variants increase.

Figure 7 shows experimental results that we have achieved a data migration rate of ~400 variants/s when the number of genotype is 2535 and annotations is 4 per variant. When processing the Chr9 file with 3484237 variants, the server costs 10110s. However, when processing the ChrX file with 3468093 variants, server costs 23781s, which is more than 2 times higher than the “chr9”. The main reason is that the number of annotations is 4 per variant in “Chr9”, while the number of annotations is 12 per variant in “ChrX”. When processing a small file ChrMT file with 3892 variants and 2534 samples, server costs 36s, approximate 100 variants/s far below the average of 400 variants/s. The reason is that VCF2JSON uses pipelining processing. The main process is responsible for reading the VCF data, a chunk of data is assigned to the multiple sub-processes after reading by main process. During the process of the multiple sub-processes, the main process continues to read the data. That is to say, the reading data and the processing data are at the same time. When the data is large enough, we can achieve more effective pipeline processing. However when the data is small, e.g., ‘ChrMT’, we cannot achieve the pipeline processing and the processing speed is also lower than average.

Table 1. Data migration results on 25 different VCF files

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Chrom | File | number of Variants | number of samples | execution time(s) |
| Chr1 | ALL.chr1.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 6332048 | 2535 | 15143 |
| Chr2 | ALL.chr2.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 6935131 | 2535 | 19963 |
| Chr3 | ALL.chr3.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 5711174 | 2535 | 16097 |
| Chr4 | ALL.chr4.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 5609535 | 2535 | 11673 |
| Chr5 | ALL.chr5.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 5203080 | 2535 | 10875 |
| Chr6 | ALL.chr6.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 4916085 | 2535 | 10202 |
| Chr7 | ALL.chr7.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 4616223 | 2535 | 10604 |
| Chr8 | ALL.chr8.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 4500423 | 2535 | 9346 |
| Chr9 | ALL.chr9.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 3484237 | 2535 | 10110 |
| Chr10 | ALL.chr10.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 3904807 | 2535 | 8072 |
| Chr11 | ALL.chr11.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 3960690 | 2535 | 8713 |
| Chr12 | ALL.chr12.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 3698195 | 2535 | 7661 |
| Chr13 | ALL.chr13.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 2795935 | 2535 | 5900 |
| Chr14 | ALL.chr14.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 2597964 | 2535 | 5365 |
| Chr15 | ALL.chr15.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 2370158 | 2535 | 4907 |
| Chr16 | ALL.chr16.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 2634262 | 2535 | 5484 |
| Chr17 | ALL.chr17.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 2275268 | 2535 | 4726 |
| Chr18 | ALL.chr18.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 2220556 | 2535 | 6360 |
| Chr19 | ALL.chr19.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 1786628 | 2535 | 3714 |
| Chr20 | ALL.chr20.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 1774522 | 2535 | 4599 |
| Chr21 | ALL.chr21.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 1077647 | 2535 | 2261 |
| Chr22 | ALL.chr22.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | 1077019 | 2535 | 2933 |
| ChrX | ALL.chrX.phase3\_shapeit2\_mvncall\_integrated.20130502.genotypes.vcf.gz | 3468093 | 2504 | 23781 |
| ChrY | ALL.chrY.phase3\_integrated\_v2a.20130502.genotypes.vcf.gz | 62042 | 1233 | 283 |
| ChrMT | ALL.chrMT.phase3\_callmom-v0\_4.20130502.genotypes.vcf.gz | 3892 | 2534 | 36 |

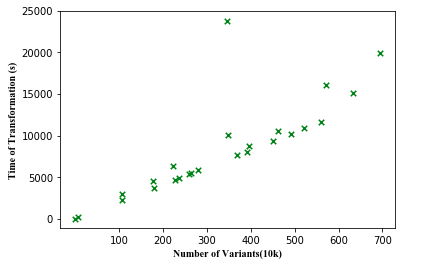


Figure 8: VCF2JSON running time in seconds when running on 25 different numbers of VCF data contained human chromosomes. All data is running on server with 20 CPUs.

To evaluate whether the data migration have data loss or not, we select four sets of files, and got the number of variants, annotations, genotype from pre-transform and post-transform files. If these elements between two files are the same, VCF2JSON does not have data loss. As shown in table2, the number of elements between VCF and JSON are the same, therefore, the data migration is achieved without loss data.

Table 2. Randomly selected four different files and get the number of elements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Files | Type | Variants | Annotationsi | genotype |
| ALL.chr1.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | .vcf.gz | 6332048 | 2719773 | 2535 |
| .json | 6332048 | 2719773 | 2535 |
| ALL.chr11.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | .vcf.gz | 3960690 | 1705573 | 2535 |
| .json | 3960690 | 1705573 | 2535 |
| ALL.chr21.phase3\_shapeit2\_mvncall\_integrated\_v2.20130502.genotypes.vcf.gz | .vcf.gz | 1077647 | 448760 | 2535 |
| .json | 1077647 | 448760 | 2535 |
| ALL.chrX.phase3\_shapeit2\_mvncall\_integrated.20130502.genotypes.vcf.gz | .vcf.gz | 3468093 | 1575038 | 2504 |
| .json | 3468093 | 1575038 | 2504 |

iAnnotations: we choose “AC=1” from INFO as the statistical standard to count how many these items in pre-transform and post-transform.

## Discussion

JSON has become a widely-used data model for massive data management, and many biomedical database applications, such as dbSNP, are transitioning to the new design by using JSON to support future massive data expansion. In addition, it is often difficulty to directly read VCF files with a large size of data, while JSON file has better readability and it is much easier to parse and understand its data. Considering the advantages of using JSON, in this paper, we develop a user-friendly mapping tool that supports the data migrations from VCF files to JSON files. VCF2JSON has a publicly available GUI for users without programming skills, and it does not run out of memory when process large size of files by reading files by chunk, due to using multiple processes to process data. VCF2JSON demonstrates its good performance through the final experiments using the files (Phase 3) in 1000 Genome Project.

## Acknowledgement

The authors thank the anonymous referees for their valuable comments and suggestions. The work was partially supported by the National Key R&D Program of China (2017YFC1200200 and 2017YFC1200205), National Natural Science Foundation of China(61602130 and 61872115), China Postdoctoral Science Foundation funded project (2015M581449 and 2016T90294), Heilongjiang Postdoctoral Fund (LBH-Z14089), Natural Science Foundation of Heilongjiang Province of China (QC2015067), and Fundamental Research Funds for the Central Universities (HIT.NSRIF.2017036)

## References

1. 1000 Genomes Project Consortium. An integrated map of genetic variation from 1,092 human genomes[J]. Nature, 2012, 491(7422): 56.
2. Müller H, Jimenez-Heredia R, Krolo A, et al. VCF. Filter: interactive prioritization of disease-linked genetic variants from sequencing data[J]. Nucleic acids research, 2017, 45(W1): W567-W572.
3. Pabinger S, Dander A, Fischer M, et al. A survey of tools for variant analysis of next-generation genome sequencing data[J]. Briefings in bioinformatics, 2014, 15(2): 256-278.
4. Medina I, De Maria A, Bleda M, et al. VARIANT: Command Line, Web service and Web interface for fast and accurate functional characterization of variants found by Next-Generation Sequencing[J]. Nucleic acids research, 2012, 40(W1): W54-W58.
5. Alemán A, Garcia-Garcia F, Salavert F, et al. A web-based interactive framework to assist in the prioritization of disease candidate genes in whole-exome sequencing studies[J]. Nucleic acids research, 2014, 42(W1): W88-W93.
6. Gosalia N, Economides A N, Dewey F E, et al. MAPPIN: a method for annotating, predicting pathogenicity and mode of inheritance for nonsynonymous variants[J]. Nucleic acids research, 2017, 45(18): 10393-10402.
7. Danecek P, Auton A, Abecasis G, et al. The variant call format and VCFtools[J]. Bioinformatics, 2011, 27(15): 2156-2158.
8. Layer R M, Kindlon N, Karczewski K J, et al. Efficient genotype compression and analysis of large genetic-variation data sets[J]. Nature methods, 2015, 13(1): 63.
9. Pedersen B S, Quinlan A R. cyvcf2: fast, flexible variant analysis with Python[J]. Bioinformatics, 2017, 33(12): 1867-1869.
10. Zheng X, Gogarten S M, Lawrence M, et al. SeqArray—a storage-efficient high-performance data format for WGS variant calls[J]. Bioinformatics, 2017, 33(15): 2251-2257.
11. Salatino S, Ramraj V. BrowseVCF: a web-based application and workflow to quickly prioritize disease-causative variants in VCF files[J]. Briefings in bioinformatics, 2016, 18(5): 774-779.
12. Hart S N, Duffy P, Quest D J, et al. VCF-Miner: GUI-based application for mining variants and annotations stored in VCF files[J]. Briefings in bioinformatics, 2015, 17(2): 346-351.
13. Teer J K, Green E D, Mullikin J C, et al. VarSifter: visualizing and analyzing exome-scale sequence variation data on a desktop computer[J]. Bioinformatics, 2011, 28(4): 599-600.
14. Gil B, Trezentos P. Impacts of data interchange formats on energy consumption and performance in smartphones[C]//Proceedings of the 2011 workshop on open source and design of communication. ACM, 2011: 1-6.
15. Jackson W. An Introduction to JSON: Concepts and Terminology[M]//JSON Quick Syntax Reference. Apress, Berkeley, CA, 2016: 15-20.
16. Barceló P, Kröll M, Pichler R, et al. Efficient Evaluation and Static Analysis for Well-Designed Pattern Trees with Projection[J]. ACM Transactions on Database Systems (TODS), 2018, 43(2): 8
17. Dede E, Govindaraju M, Gunter D, et al. Performance evaluation of a mongodb and hadoop platform for scientific data analysis[C]//Proceedings of the 4th ACM workshop on Scientific cloud computing. ACM, 2013: 13-20.
18. Pandey M, Pandey R. JSON and its use in Semantic Web[J]. International Journal of Computer Applications, 2017, 164(11)
19. Storey V C, Song I Y. Big data technologies and management: What conceptual modeling can do[J]. Data & Knowledge Engineering, 2017, 108: 50-67.
20. Penha E D S, Iriabho E, Dussaq A, et al. Isomorphic semantic mapping of variant call format (VCF2RDF)[J]. Bioinformatics, 2016, 33(4): 547-548.
21. Li H. BGT: efficient and flexible genotype query across many samples[J]. Bioinformatics, 2015, 32(4): 590-592.
22. Kelleher J, Ness R W, Halligan D L. Processing genome scale tabular data with wormtable[J]. BMC bioinformatics, 2013, 14(1): 356.
23. Hsu K C, Leu J S. Improving the efficiency of presence service in IMS by JSON[C]//Ubiquitous and Future Networks (ICUFN), 2015 Seventh International Conference on. IEEE, 2015: 547-550.
24. Singh S, Rawat M. Design and Analysis of Visual Insight of Child using IBM Multimedia Analysis and Retrieval System[J]. International Journal of Computer Applications, 2017, 161(6).
25. Izquierdo J L C, Cabot J. JSONDiscoverer: Visualizing the schema lurking behind JSON documents[J]. Knowledge-Based Systems, 2016, 103: 52-55.
26. Carpov S, Tortech T. Secure top most significant genome variants search: iDASH 2017 competition[J]. BMC Medical Genomics, 2018, 11(4): 82.
27. Chang X, Wang K. wANNOVAR: annotating genetic variants for personal genomes via the web[J]. Journal of medical genetics, 2012: jmedgenet-2012-100918.
28. Deng J E, Sham P C, Li M X. SNPTracker: a swift tool for comprehensive tracking and unifying dbSNP rs IDs and genomic coordinates of massive sequence variants[J]. G3: Genes, Genomes, Genetics, 2016, 6(1): 205-207.