We would like to thank the editor for a very thoughtful and detailed review of our manuscript. Incorporation of the editor's suggestions has led to a much improved manuscript. Below we provide a point-by-point response to the editor's comments and how we have addressed them in the revised manuscript.

[Comment]: cite the manuscript of Palmera et al., 2011 (and maybe others) "stressing the importance of utilizing data aggregation to achieve high bandwidth" as motivation of your work at a suited position in your text (maybe introduction).

[Response]: We have added Palmera's paper as a reference (on page 13 line 6 -8) and briefly mentioned on page 2 line 27 – 28 why it is relevant to our work, namely data aggregation.

[Comment]: add references for the netcdf and pnetcdf libraries (at least the URLs, or even better papers, if available).

[Response]: We have included the URLs for netCDF (on page 3 line 24) and pnetCDF (on page 3 line 2) as well as a paper that describes pnetCDF as a reference (on page 3 line 1 and on page 12 line 2 - 3).

[Comment]: explicitely mention (e.g., in Section 3) which type of pnetcdf interface you used: collective parallel netcdf API. (see your reply on page C3216)

[Response]: As suggested, we now explicitly mention we used collective parallel netCDF API in the revised manuscript on page 5 line 11 - 12.

[Comment]: add a discussion section, where you elaborate on important topics: a) The issue on I/O performance tuning (with reference to Behzad and Lu, 2013) for application scientits and the limitation that you did not use such tuning (stripe count, stripe size) (see comment/reply on page C3219/C3220). This limitation needs to be explicitely mentioned. b) Elaborate a bit on the pros and cons and the limitations of application level versus software stack level data aggregation techniques (comments/replies on pages C3215, C3220/C3221)

[Response]: We have added Behzad and Luu, 2013 paper as a reference in the revised manuscript. Automated I/O performance tuning is a great tool that deals with all layers in the I/O stack. In fact we had experimented with stripe count and stripe size in the parallel file system layer in our study. The objective is to look for an "optimal" setting for those two parameters to maximize the utilization of the parallel file system. Behzad and Luu's paper indicates adjusting these two parameters has a larger impact on performance than other parameters in the I/O stack. This tool will be more suitable for production environment since the simulated domain is fixed. However, it might not be suitable in cases where model applications are evolving requiring corresponding domain changes. In such scenarios it takes quite a long time to run such tool as the authors stated.

We don't think you can compare application level and stack level data aggregation with pros and cons since they represent two different perspective and they can co-exist rather than one having choose one

or the other. Application level data aggregation provides benefits in larger chunk of data going to the disk which in turns translates into high I/O rate, as well as higher degree of contiguousness in data which translates into faster write time due to less number of seek operations. To our knowledge, data aggregation in the I/O stack level deals with aggregation of I/O requests. Hence with the implementation of the application level data aggregation in a scientific model, such model can still utilize the automated I/O performance tuning tool. If there is any additional performance improvement, it will be coming from the other I/O stack level other than the parallel file system since stripe count and stripe were considered in our approach.

With this explanation, we have added the following paragraph in the manuscript in Section 2 on page 3 line 27 -32:

An auto-tuning framework (Behzad and Luu et al., 2013) has been developed to attempt to provide the best I/O setting with respect to the entire I/O stack, automatically. It used a genetic algorithm to search for the “optimal” solution from the parameter-space. Our approach gears toward application level rather than the I/O stack level but we also dealing with two parameters, stripe count and stripe size in the parallel file system level.

[Comment]: state, e.g. in Section 4.2 that the simplified code is available on request

[Response]: We have added “and is available on request” on the manuscript in Section 4.2 on page 6 line 21.

[Comment]: Moreover, I think there was a misunderstanding of one referee comment (see page C3215, last paragraph and subsequent page). As far as I understand the referee an argument on your text is oversimplified. The referee states:

"In section 5.3 the authors claim that increasing the data size on the processor which is responsible for I/O will translate into higher I/O rates. If this is the case why not just use the original serial approach in which one I/O processor is responsible for all of the data?"

So, please elaborate on the question, why this simple connection (increased data size -> higher I/O rate) does obviously not hold completely (otherwise serial I/O should be the fastest). Please modify your text/argumentation accordingly.

[Response]: Thanks for the comments. To clarify this issue, we listed what we explained in the previous response (the two paragraphs below) and added more explanations in the last paragraph.

“In our approach, the process consists of two parts: data aggregation (communication time) and data write to disk (I/O time) by a subset of processors. So the sum of the communication time and the I/O time is the true representation of time to move data in each processor to the disk. This timing can be used to compare directly with the parallel I/O approach implemented with pnetCDF which sends data to the disk collectively without any communication.

Figure 11 illustrates the notion that larger chunks of data have better I/O rates. We argue that the benefit of data aggregation is the basis for the success of our approach. We won't go to the extreme to have one processor to do the I/O (this is the technique being employed in the current CMAQ model) but rather try to strike for a balance by utilizing parallel I/O technology. Indeed, in this article, we have demonstrated our approach out performs the serial approach which is currently used in the CMAQ model, substantially."

We attempt to address this from a different perspective. First of all, it is known that the I/O rate to write a 64MB of data as an example to the disk is higher than to write a 24KB of data. This fact reflects in Figure 11. Furthermore, the time it takes to write a 64MB of data to disk is longer than to write a 24KB of data. Increase data size result in higher I/O rate does not imply serial I/O performs faster because 1. in order to do I/O in serial mode, a data gathering process needs to be performed which takes a substantial amount of time, 2. writing an entire data takes longer time than to write n sub-chunks of the original data in parallel using parallel I/O library. These two arguments are supported by our experiment and the timing data we gathered.